VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

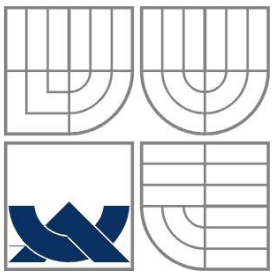
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYHLEDÁVÁNÍ INFORMACÍ V DIGITÁLNÍCH KNIHOVNÁCH

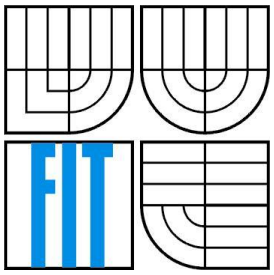
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. PETR HOCHMAL
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VYHLEDÁVÁNÍ INFORMACÍ V DIGITÁLNÍCH KNIHOVNÁCH

DIGITAL LIBRARY INFORMATION RETRIEVAL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. PETR HOCHMAL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. PETR CHMELAŘ

BRNO 2010

Zadání diplomové práce/9388/2009/xhochm01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Hochmal Petr, Bc.**
Obor: Informační systémy
Téma: **Vyhledávání informací v digitálních knihovnách**
Digital Library Information Retrieval
Kategorie: Databáze

Pokyny:

1. Seznamte se s problematikou vyhledávání informací.
2. Zaměřte se na textová data, jejich analýzu, sémantické slovníky, případně na jejich kombinaci.
3. Nalezněte a pokuste se vyřešit aktuální problémy v oblasti vyhledávání informací dle pokynů vedoucího.
4. Navrhněte a vytvořte systém pro uchovávání, správu a dotazování zvolených dat pro demonstraci navrženého přístupu na příkladě elektronické knihovny.
5. Zhodnoťte vlastnosti a případná vylepšení demonstrační aplikace.

Literatura:

- BAEZA-YATES, Ricardo - RIBEIRO-NETO, Berthier. *Modern information retrieval*. New York: ACM Press, 1999. 513 p. ISBN 0-201-39829-X.
- KOSCH, Harald. *Distributed Multimedia Database Technologies Supported by MPEG-7 and MPEG-21*. CRC Press, 2003. ISBN 0-8493-1854-8.
- CHMELAŘ, Petr. Multimediální databáze. 59 s. 2006.

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Chmelař Petr, Ing.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá způsoby vyhledávání informací. Popisuje modely pro vyhledávání dat a metody pro hodnocení efektivity systémů pro vyhledávání informací. Přibližuje principy zpracování vstupních textů pro IR systémy s použitím seznamu stop slov a stemmeru. Dále ukazuje způsob rozšíření dotazů o synonyma pomocí thesauru, metody pro zohlednění frází v dotazech a představuje myšlenku ohodnocení dokumentu dle stupně podobnosti výskytu fráze. V druhé polovině práce je popsán návrh systému pro vyhledávání s užitím vektorového modelu, rozšíření dotazu o synonyma a zohlednění výskytů frází. Tento systém je implementován v jazyce C# jako aplikace pro vyhledávání a správu dokumentů v digitálních knihovnách. Efektivita tohoto systému je pomocí několika testů vyhodnocena na konci práce.

Klíčová slova

Vyhledávání informací, vektorový model, relevance, měření efektivity vyhledávání, slovník synonym, analýza textu, stemming, vyhledávání frází, .NET aplikace, SOA.

Abstract

This thesis deals with methods of information retrieval. Firstly, it describes models of information retrieval and methods of retrieval evaluation. Then it brings closer the principles of the input text processing for IR with use of stopword list and stemmer. Furthermore, it shows the way of the query expansion with synonyms using the thesaurus, methods of handling phrases appearance in queries and introduces the idea of ranking documents by the degree of phrase occurrence similarity in documents. In the second part of this thesis is described the design of whole IR system with using vector model, query expansion with synonyms and phrases handling. This system has been implemented in C# as the application for retrieving and administration of the documents in digital libraries. The effectiveness of this system has been evaluated at the end of this thesis by several tests.

Keywords

Information retrieval, vector space model, relevance, retrieval evaluation, thesaurus, text analysis, stemming, phrase retrieval, .NET application, SOA.

Citace

Petr Hochmal: Vyhledávání informací v digitálních knihovnách, diplomová práce, Brno, FIT VUT v Brně, 2010

Vyhledávání informací v digitálních knihovnách

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Hochmal
25. 5. 2010

Poděkování

Tímto bych rád poděkoval vedoucímu práce Ing. Petru Chmelařovi, za jeho odbornou pomoc a čas, který mi věnoval při vytváření tohoto díla. Dále také své rodině za podporu při studiu.

Obsah

1 Úvod

V dnešní době existuje nejen na internetu obrovské množství textových dat, ve kterých je bez efektivního vyhledávacího systému téměř nemožné naleznout ty informace, které nás zajímají, tedy jsou pro nás relevantní. Obor zabývající se problematikou strojového vyhledávání informací zaměstnává odborníky již několik desítek let, a za tu dobu se posunul od katalogových až po fulltextové vyhledávače. Vznikla řada modelů, které popisují metody vyhledávání dokumentů, jmenovitě například vektorový nebo pravděpodobnostní. Tyto modely však neuvažují sousednost slov v textu a tak dochází ke ztrátě přesnosti při určování relevance dokumentů vůči dotazu. Jedná se o problematiku vyhledávání informací zohledňující sousednost slov v dokumentu, tedy výskyt frází. Dalším způsobem, jakým by se dalo zlepšit nalézání relevantních dokumentů je vyhledání informací, které nejsou přímo dané dotazem, například vyjádřena synonymy.

Tato práce si klade za cíl navrhnout a implementovat systém pro efektivní vyhledávání informací za použití některých ze způsobů, které by mohly vést k lepšímu ohodnocení dokumentů. Jmenovitě se jedná právě o rozšíření dotazu o synonyma a problematiku vyhledávání frází. Ten by měl umožňovat vyhledat pomocí jednoduchého dotazu dokumenty, které obsahují i jinak formulovanou, ale požadovanou informaci. Výsledkem tohoto snažení má být aplikace pro vyhledávání informací v elektronických knihovnách, která bude umožňovat správu dokumentů a vyhledávání v nich.

1.1 Struktura práce

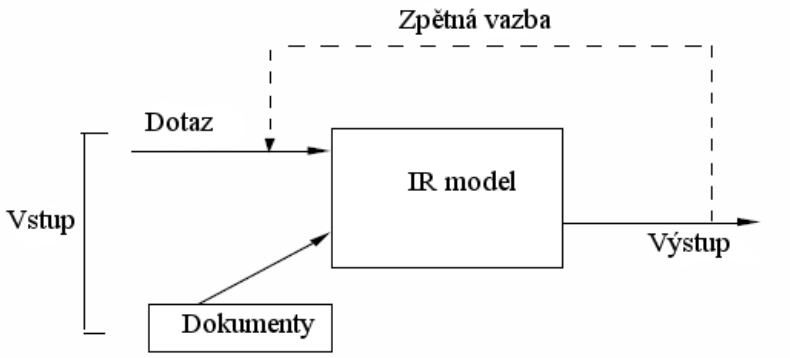
Cílem práce je vytvořit systém pro správu dokumentů a vyhledávání informací v nich. Ten by měl umožňovat efektivní vyhledávání v dokumentech s využitím navržených technik.
Tato práce navazuje na stejnojmenný semestrální projekt, který se zabýval teoretickým úvodem do problematiky vyhledávání informací v textech. Z této práce převzaté kapitoly, konkrétně teoretické rozbory modelů pro vyhledávání, popis metrik pro měření výkonnosti a způsoby zpracování textů, byly aktualizovány ve smyslu upřesnění uvedených informací. Jedná se o kapitoly 2, 3 a 4 této práce.

Kapitola 2 seznamuje s modely pro vyhledávání informací s důrazem na vektorový model, se kterým se pracuje dále. Poté se zabývá způsoby zvýšení efektivity vyhledávání z pohledu zohlednění sousednosti slov, respektive výskytu frází, které čistý vektorový model neumožňuje, a představuje návrh modifikace ohodnocení dokumentu dle podobnosti výskytu fráze. Ve třetí kapitole jsou uvedeny metriky pro hodnocení výkonnosti systémů pro vyhledávání informací. Čtvrtá kapitola uvádí do problematiky zpracování textu, které je potřeba pro získání vlastností jak dokumentů, tak dotazů. Zaměřuje se na tokenizaci, tedy rozdělení textu na kandidáty na indexová slova, eliminaci častých slov a jejich úpravu na základní tvar. Dále představuje možnosti využití slovníku synonym pro možné zvýšení efektivity vyhledávání. Pátá kapitola se zabývá specifikací a návrhem systému pro vyhledávání, který využívá vektorového modelu a metod pro zlepšení ohodnocení dokumentů prostřednictvím slovníků synonym a ohodnocení dle podobného výskytu frází. V šesté kapitole je popsána implementace navržené aplikace. Sedmá kapitola popisuje experimenty, týkající se převážně efektivity vyhledávání, které proběhly na implementované aplikaci. Zároveň s tím jsou zhodnoceny výsledky uskutečněných testů.

2 Modely pro vyhledávání informací

Modely pro vyhledávání informací (dále jen IR modely) představují strukturu informací (z dokumentů) a procesy nad nimi, které vedou k získání znalosti o relevantnosti dokumentů, nad kterými vyhledáváme. Cílem vyhledávání je tedy vrátit uživateli na jeho dotaz seznam dokumentů, které odpovídají jeho informační potřebě. V tomto ohledu je dobré zmínit vyhledávání dat, které se zaměřuje na přesné uspokojení dotazu, to znamená nalezení dokumentů (zdrojů), které splňují exaktně podmínku obsahu dotazovaných dat.

V systému pro vyhledávání informací by se dal model pro vyhledávání informací zobrazit dle Obr. 2.1. Systém se skládá ze vstupu, jehož součástí je množina dokumentů, nad kterými se vyhledává, a dotazu, který specifikuje informační potřebu. Dále ze samotného IR modelu, výstupu, jež obsahuje množinu relevantních dokumentů, a volitelně zpětné vazby, která umožňuje uživateli pomocí výstupu specifikovat dotaz, čímž může dále zlepšovat výsledky vyhledávání.



Obr. 2.1: Systém pro vyhledávání informací (Zdroj [5])

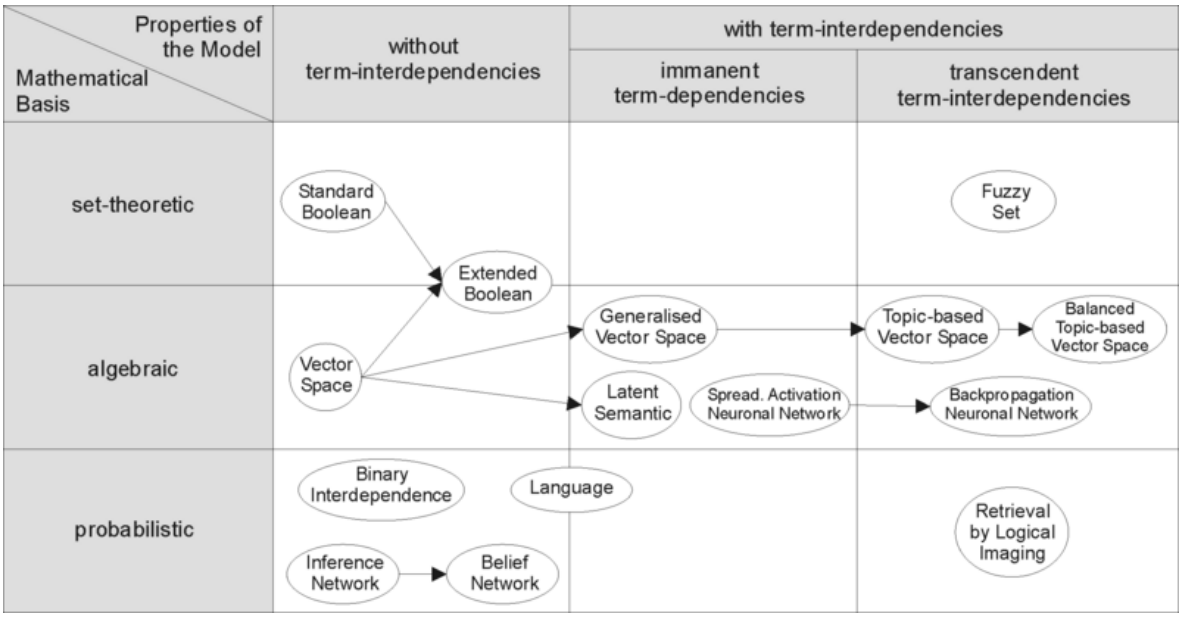
Modely pro vyhledávání informací se dle základního principu fungování rozdělují do tří skupin [1]:

- Množinové
- Algebraické
- Pravděpodobnostní

Základními zástupci těchto skupin jsou booleovský model, pracující s booleovskou algebrou nad množinami slov, vektorový model, kde je text reprezentován n-rozměrným vektorem, a pravděpodobnostní model (známý jako model binárně nezávislého vyhledávání), pracující s teorií pravděpodobnosti. V následujících podkapitolách budou rozebrány tyto základní modely podrobněji. Další modely je možno vidět na Obr. 2.2. Jedná se převážně o modely, které berou v potaz imanentní nebo transcendentní závislost slov.

Předem by bylo vhodné poznamenat, že výše zmíněné modely (booleovský a vektorový) jsou vhodné spíše jako teoretický základ pro další úpravy s ohledem na jejich efektivnost vyhledávání (booleovský model) nebo výpočetní náročnost (vektorový model – práce s n-dimenzionálními vektory).

Jako zdroj pro popis modelů posloužil [1] – kapitola 2. V dalších kapitolách budeme uvažovat a pracovat s vektorovým modelem, nebude-li zmíněno jinak.



Obr. 2.2: Modely pro vyhledávání informací (Zdroj [8])

Obecné principy

Před seznámením s konkrétními IR modely bude formálně charakterizován obecný IR model [1].

Model pro vyhledávání informací je čtveřice $\{\boldsymbol{D}, \boldsymbol{Q}, F, R(q_i, d_j)\}$, kde:

- ***D*** *je množina složená z logických pohledů reprezentující dokumenty.*
- ***Q*** *je množina složená z logických pohledů reprezentující uživatelský dotaz.*
- *F je systém pro modelování dokumentů, dotazů a jejich vztahů*
- $R(q_i, d_j)$ *je hodnotící (ranking) funkce, která dvojici* $q_i \in D$ *a* $d_i \in Q$ *přiřadí reálné číslo. Tedy ranking funkce definuje uspořádání dokumentů s ohledem na dotaz.*

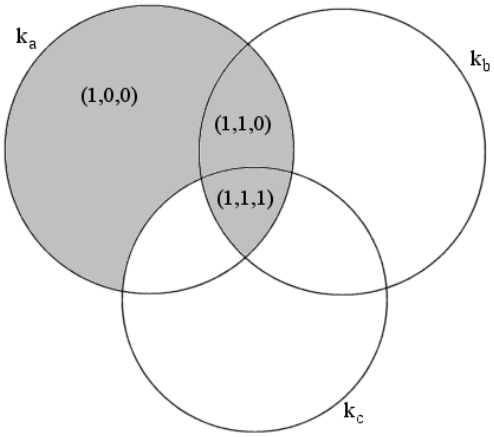
V IR modelech je každý dokument popsán množinou reprezentativních klíčových slov, která se nazývají indexová slova (index terms). Tato slova by měla být schopna svojí sémantikou popsat samotný dokument. Obecně vzato, ne všechna slova mají stejnou vypovídací hodnotu o dokumentu. Proto je dobré indexovým slovům přiřadit různou váhu pro reprezentaci dokumentu.

Nechť k_i je indexové slovo, d_j je dokument, pak $w_{i,j} \geq 0$ je váha přiřazená páru (k_j, d_j). Máme tedy množinu $K=\{k_1,\ldots,k_n\}$ obsahující všechna klíčová slova. Váha $w_{i,j}$ je pak přiřazena každému dokumentu pro každé indexové slovo. Pro ta indexová slova, která se nevyskytují v dokumentu, je $w_{i,j} = 0$.

2.1 Booleovský model

Tento model je jeden z nejstarších používaných pro vyhledávání informací. Jeho základem je teorie množin a booleovská algebra. Dotazy jsou dány booleovskými výrazy, které jsou aplikovány na každý dokument. Jako výsledek na dotaz vrací pouze ty dokumenty, které přesně odpovídají

položenému dotazu. To je způsobeno samotným principem booleovské algebry. Neumožňuje tedy nalézt částečnou shodu. Jedná se tak spíše o model pro vyhledávání dat, kde je tato vlastnost žádoucí.



Obr. 2.3: Tři konjunktivní komponenty pro dotaz $[\ \boldsymbol{q = k_a \wedge (k_b \vee \neg\, k_c)}\ ]$

Váhy indexových slov jsou binární, tedy $w \in \{0,1\}$. Dotaz je tvořen indexovými slovy a operátory *and*, *or* a *not*. Je ho tedy možné transformovat na disjunkci konjunktivních vektorů. Např. dotaz $[\ \boldsymbol{q = k_a \wedge (k_b \vee \neg\, k_c)}\ ]$ může být zapsán jako $\left[\ \vec{q}_{dnf} = (1,1,1) \vee (1,1,0) \vee (1,0,0)\ \right]$ viz Obr. 2.3 [1].

Hlavní výhodou booleovského modelu je jeho jednoduchost, nevýhodou je však jeho neschopnost pracovat s jemnějšími vahami indexových slov.

2.2 Pravděpodobnostní model

Tento model se snaží zachytit problém vyhledávání informací pomocí teorie pravděpodobnosti. Myšlenka je následující. Na daný dotaz existuje množina dokumentů, obsahující právě ty dokumenty, které jsou relevantní, a žádné jiné. Stanovme si tuto množinu dokumentů jako ideální odpověď. Pokud bychom měli přesný popis této odpovědi, nebyl by problém získat relevantní dokumenty. Můžeme tedy uvažovat proces dotazování jako proces hledání popisu ideální odpovědi. Úkolem je tedy vytvořit počáteční odhad množiny relevantních dokumentů R a uživatelskou interakcí zlepšovat popis ideální odpovědní množiny. Dokumenty mimo množinu R jsou považovány za nerelevantní.

Pro daný dotaz q přiřadí pravděpodobnostní model každému dokumentu d_j, jako měřítko jeho podobnosti k dotazu, poměr $P(d_j \text{ relevantní k } q)/P(d_j \text{ není relevantní k } q)$, který určí šanci dokumentu d_j být relevantní pro dotaz q.

V pravděpodobnostním modelu jsou váhy všech indexových slov binární (např. $w_{i,j} \in \{0,1\}$, $w_{i,q} \in \{0,1\}$). Dotaz q je podmnožina indexových slov. Dále víme, že R je množina relevantních (nebo z počátku považovaných za relevantní) dokumentů. $\bar{R}$ je doplňkem této množiny, tedy množinou nerelevantních dokumentů. Nechť je $P\left(R|\overrightarrow{d_j}\right)$ pravděpodobnost, že je dokument d_j relevantní k dotazu q, a $P(\overline{R}|\overrightarrow{d_j})$ pravděpodobnost, že dokument d_j není relevantní k dotazu q. Pak je podobnostní funkce $sim(d_j, q)$ pro dokument d_j a dotaz q vyjádřena následovně (2.1):

$$sim(d_j, q) = \frac{P(R|\vec{d_j})}{P(\overline{R}|\vec{d_j})} \tag{2.1}$$

S použitím Bayesova pravidla:

$$sim(d_j, q) = \frac{P(\vec{d_j}|R) \times P(R)}{P(\vec{d_j}|\overline{R}) \times P(\overline{R})} \tag{2.2}$$

$P(\vec{d_j}|R)$ vyjadřuje pravděpodobnost náhodného vybrání dokumentu d_j z množiny relevantních dokumentů R. Dále $P(R)$ vyjadřuje pravděpodobnost, že dokument náhodně vybraný z množiny všech dokumentů je relevantní. Význam $P(\vec{d_j}|\overline{R})$ a $P(\overline{R})$ je analogický a doplňkový k výše popsanému.

Vzhledem k tomu, že $P(\overline{R})$ a $P(R)$ je pro všechny dokumenty stejný, můžeme podobnostní funkci zapsat jako:

$$sim(d_j, q) \sim \frac{P(\vec{d_j}|R)}{P(\vec{d_j}|\overline{R})} \tag{2.3}$$

Při předpokladu nezávislosti indexových slov, zapíšeme podobnost následovně:

$$sim(d_j, q) \sim \frac{(\prod_{g_i(\vec{d_j})=1} P(k_i|R)) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k_i}|R))}{(\prod_{g_i(\vec{d_j})=1} P(k_i|\overline{R})) \times (\prod_{g_i(\vec{d_j})=0} P(\overline{k_i}|\overline{R}))} \tag{2.4}$$

kde $P(k_i|R)$ vyjadřuje pravděpodobnost, že se indexové slovo k_i vyskytuje v dokumentu náhodně vybraném z množiny R. $P(\overline{k_i}|R)$ vyjadřuje pravděpodobnost, že indexové slovo k_i nevyskytuje v dokumentu náhodně vybraném z množiny R. Významy zbylých pravděpodobností spojených s množinou $\overline{R}$ jsou opět analogické k výše popsanému.

Použitím logaritmu, faktu, že $P(k_i|R) + P(\overline{k_i}|R) = 1$ a ignorováním faktorů, které jsou v rámci jednoho dotazu vždy stejné, zapíšeme rovnici jako:

$$sim(d_j, q) \sim \sum_{i=1}^{t} w_{i,q} \times w_{i,j} \times \left(\log \frac{P(k_i|R)}{1 - P(k_i|R)} + \log \frac{1 - P(k_i|\overline{R})}{P(k_i|\overline{R})} \right) \tag{2.5}$$

Tím získáme hlavní výraz pro výpočet ohodnocení v pravděpodobnostním modelu.

Vzhledem k tomu, že na počátku není známa množina R, je nutné stanovit metody pro počáteční výpočet pravděpodobností $P(k_i|R)$ a $P(k_i|\overline{R})$. Na úplném začátku, hned po specifikaci dotazu, nejsou žádné nalezené dokumenty. Musí být tedy stanoveny zjednodušující předpoklady (2.6): $P(k_i|R)$ je konstantní pro všechna indexová slova a předpokládejme, že rozdělení indexových slov napříč nerelevantními dokumenty může být aproximováno jejich rozdělením mezi všechny dokumenty v kolekci.

$$\begin{aligned} P(k_i|R) &= 0{,}5 \\ P(k_i|\overline{R}) &= \frac{n_i}{N} \end{aligned} \tag{2.6}$$

Kde n_i je počet dokumentů obsahujících indexové slovo i a N je počet všech dokumentů v kolekci. Díky tomuto odhadu lze získat dokumenty, které obsahují indexová slova z dotazu, a dodat jim pravděpodobnostní ohodnocení.

Poté je počáteční ohodnocení modifikováno následovně. Definujme V jako podmnožinu počátečně nalezených dokumentů (například prvních r dokumentů, kdy r bude předdefinovaná mez). Dále V_i bude podmnožina z V, složená z dokumentů obsahujících indexové slovo k_i. Pro vylepšení pravděpodobnostního ohodnocení musíme vylepšit (2.7) odhady pro $P(k_i|R)$ a $P(\bar{k}_i|R)$. To provedeme následovně: Můžeme upřesnit $P(k_i|R)$ rozložením indexového slova k_i napříč dosud nalezenými dokumenty a upřesnit $P(k_i|\bar{R})$ s faktem, že všechny nenalezené dokumenty jsou nerelevantní.

$$P(k_i|R) = \frac{|V_i|}{|V|}$$
$$P(k_i|\bar{R}) = \frac{n_i - |V_i|}{N - |V|} \tag{2.7}$$

Toto můžeme opakovat rekurzivně. Tím jsme schopni vylepšit odhady pravděpodobností $P(k_i|R)$ a $P(\bar{k}_i|R)$, aniž by probíhala interakce s uživatelem.

Poslední vzorce pro stanovení $P(k_i|R)$ a $P(\bar{k}_i|R)$ řeší v praxi vznikající problém s nízkými hodnotami $|V|$ a $|V_i|$ (například $|V| = 1$ a $|V_i| = 0$). Pro vypořádání se s tímto problémem se obvykle přidává korekční činitel, tedy:

$$P(k_i|R) = \frac{|V_i| + 0{,}5}{|V| + 1}$$
$$P(k_i|\bar{R}) = \frac{n_i - |V_i| + 0{,}5}{N - |V| + 1} \tag{2.8}$$

Použití korekčního činitele, který je konstantní, není vždy uspokojující. Alternativou je použít podíl n_i/N . Tím získáme následující rovnice pro pravděpodobnosti.

$$P(k_i|R) = \frac{|V_i| + \frac{n_i}{N}}{|V| + 1}$$
$$P(k_i|\bar{R}) = \frac{n_i - |V_i| + \frac{n_i}{N}}{N - |V| + 1} \tag{2.9}$$

Hlavní výhodou pravděpodobnostního modelu pro vyhledávání informací je to, že dokumenty jsou ohodnoceny v sestupném pořadí dle jejich pravděpodobnosti být relevantní. Mezi nevýhody patří nutnost počátečního odhadu rozdělení dokumentů do množin relevantních a nerelevantních, dále skutečnost, že není brán v potaz počet výskytů indexového slova v dokumentu a v poslední řadě je nevýhodou nutnost přijetí předpokladu nezávislosti indexových slov.

2.3 Vektorový model

U předešlého booleovského modelu jsme uvedli, že binární váhy jsou příliš limitujícím faktorem. Tuto nectnost řeší vektorový model díky svému jemnému výpočtu vah, respektive způsobu určování podobnosti dokumentů. Jak bylo výše zmíněno, každé indexové slovo má přiřazenu váhu. Ve vektorovém modelu je každý dokument, nebo dotaz reprezentován t-rozměrným vektorem, jehož dimenze jsou složeny z vah všech indexových slov ($t = |K|$). Tyto vektory jsou využity k výpočtu *stupně podobnosti* dokumentu a dotazu (nebo jiného dokumentu). Díky tomuto je možné získat i částečnou shodu ve vyhledávání a podle stupně podobnosti řadit výsledky vyhledávání s ohledem na relevanci.

Shrnutím dosavadních poznatků získáme:

- Váhu $w_{i,j}$ přiřazenou dvojici (k_i, d_j), $w_{i,j} \in \mathbb{R}^{0+}$
- Váhu $w_{i,q}$ přiřazenou dvojici (k_i, q), $w_{i,q} \in \mathbb{R}^{0+}$
- Vektor $\vec{q} = (w_{1,q}, w_{2,q}, \dots, w_{t,q}), t = |K|$
- Vektor $\overrightarrow{d_j} = (w_{1,j}, w_{2,j}, \dots, w_{t,j}), t = |K|$

Vektory $\vec{q}$ a $\overrightarrow{d_j}$ budou použity pro výpočet stupně podobnosti mezi dotazem q a dokumentem d_j. K tomuto lze použít výpočet kosinu úhlu mezi těmito dvěma vektory [7],

$$sim(d_j, q) = \frac{\overrightarrow{d_j} \cdot \vec{q}}{|\overrightarrow{d_j}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,q}^2}} \tag{2.10}$$

Místo rozhodování o relevantnosti dokumentu, vektorový model určuje stupeň podobnosti dokumentu a dotazu. Můžeme tedy určit mez, nad kterou budou dokumenty hodnoceny dle podobnosti jako relevantní. Nejprve je nutné definovat výpočet vah pro indexová slova, který v podstatě určuje míru chování modelu.

V nejjednodušším případě může být váha $w_{i,j}$ rovna frekvenci slova *i* v dokumentu *j*. Ovšem přesnější je použití normované frekvence,

$$ntf_{i,j} = \frac{freq_{i,j}}{max_l\, freq_{l,j}} \tag{2.11}$$

kde $freq_{i,j}$ je frekvence *i*-tého indexového slova v *j*-tém dokumentu a *max* určuje nejfrekventovanější slovo, tím získáme frekvenci slova v intervalu <*0,1*>nezávisle na jeho počtu výskytů. V praxi se spíše používá následující modifikace výpočtu frekvence slova [2] s použitím koeficientu $a \in \langle 0, 1)$,

$$ntf_{i,j} = a + (1 - a)\frac{freq_{i,j}}{max_l\, freq_{l,j}} \tag{2.12}$$

který vyzdvihuje slova obsažená v dokumentu od těch nevyskytujících se. Pak tedy platí:

1. $w_{i,j} = 0, pro\ k_i \notin d_j$
2. $w_{i,j} \in < a, 1 >, pro\ k_i \in d_j$

Váhu indexového slova dále ovlivňuje výskyt tohoto slova napříč dokumenty, tedy slovo, jež je obsaženo pouze v jednom nebo několika málo dokumentech, je cenné. Naopak slovo vyskytující se ve všech dokumentech nemá o jednom konkrétním dokumentu žádnou vypovídací váhu. K tomuto nám slouží takzvaná inverzní dokumentová frekvence slova. Její výpočet je definován rovnicí

$$idf_i = \log \frac{N}{n_i} \tag{2.13}$$

kde N je počet všech dokumentů a n_i je počet dokumentů obsahujících indexové slovo k_i.

Dostáváme se tedy k celkovému výpočtu vah, a to:

$$w_{i,j} = a + (1-a) \frac{freq_{i,j}}{max_l\, freq_{l,j}} \times \log \frac{N}{n_i} \tag{2.14}$$

Jak již ze složení rovnice vyplývá, tato strategie výpočtu vah se nazývá tf-idf schéma.

Pro výpočet vah se využívá stejného principu. Dle [1] je pro dobré výsledky doporučeno použití koeficientu *a=0.5*:

$$w_{i,j} = 0.5 + \frac{0.5\, freq_{i,j}}{max_l\, freq_{l,j}} \times \log \frac{N}{n_i}$$

Hlavní výhodou vektorového modelu je jeho schéma výpočtu vah indexových slov, které zvyšuje schopnosti vyhledávání, dále možnost částečné shody ve vyhledávání s dotazem a určování podobnosti dokumentů, umožňující ve výsledku řadit výsledky hledání dle získaného skóre.

Nevýhodou je, že indexová slova v dokumentech jsou uvažována jako vzájemně nezávislá, nelze tedy zohlednit sousednost slov.

2.4 Vyhledávání se zohledněním frází

Uživatelé obvykle pro vyhledávání používají slovní spojení (fráze), které očekávají, že bude nalezený dokument obsahovat. Uvažujeme-li použití vektorového modelu pro vyhledávání, víme, že se při zjišťování podobnosti dokumentu a dotazu nezohledňuje sousednost slov. V této kapitole si představíme metody, které by mohly vykazovat dobré výsledky při vyhledávání pomocí dotazu obsahujícím fráze. Zajímavý způsob zohlednění vztahů slov představuje dokument [6], kde je závislost mezi indexovými slovy zohledněna pomocí asociačních pravidel.

2.4.1 N-gramy slov

Jedním z poměrně jednoduchých způsobů, jak zohlednit sousednost slov a tím i napomoci vyhledávání frází je použití n-gramů slov. Jiný výraz pro n-gram je v angličtině n-words (například biwords, triwords, atp.). Princip je takový, že v průběhu indexace dokumentů nebudeme uvažovat,

že *1 token = 1 indexové slovo*, ale indexové slovo bude složeno z n-tice po sobě jdoucích tokenů. Dle [2] již použití tri-gramu vykazuje dobré výsledky při vyhledávání frází.

Jako příklad si uveďme dotaz *„fit vut brno“*. Při použití bigramů rozdělíme dotaz na dvojice:

- *„fit vut“* a
- *„vut brno“*

a tyto dvojice následně použijeme jako indexová slova pro vyhledávání. Stále je sice šance získání falešných pozitiv (s ohledem na celou frázi), ale jak bylo poznamenáno výše, při použití trigramů to není častý jev.

Problémem však je, že použití n-gramů velmi zvyšuje velikost slovníku indexových slov. Navíc při indexovém slovu složeném z např. trigramu není jedno slovo přirozeně zpracováno (bylo by nutné ho vyhledávat v každém trigramu) a tak je stále potřeba udržovat v indexu jednotlivá slova. Dále abychom mohli zohlednit dvojslovné fráze, je třeba udržovat také v slovníku indexových slov bigramy.

Při zpracování dotazu můžeme uvažovat 2 způsoby frázování – implicitní a explicitní. Při použití *explicitních* frází uživatel vymezí fráze například uvozovkami. Poté slova mimo uvozovky uvažujeme jako jednoduchá indexová slova a uvnitř uvozovek je rozdělujeme na co největší používané n-gramy). V případě implicitního frázování rozdělíme dotaz na 1 až n-tice slov a ty poté najednou použijeme pro vyhledávání. V tomto případě by bylo vhodné uvažovat vyšší váhu indexového slova se zvyšujícím se stupněm n-gramu, aby dokumenty obsahující fráze získaly vyšší váhu v množině odpovědí.

2.4.2 Poziční indexy

Vektorový model vyhledávání pracuje pouze s informací, ve kterých dokumentech se indexové slovo vyskytuje. Abychom ale mohli zohlednit sousednost nebo lépe vzdálenost slov v dokumentu, je třeba si uchovávat pro každou dvojici *{indexové slovo, dokument}* pozice výskytů indexového slova. Nazvěme tyto seznamy pozičními indexy.

Myšlenka použití pozičních indexů je následující. Uvážíme-li, že uživatel svůj dotaz píše takovým stylem, jak očekává, že se bude vyskytovat v textu, měli bychom stupeň podobnosti dokumentu s dotazem ovlivnit také podobností vztahu slov dotazu se slovy v dokumentech. Tedy brát v úvahu vzdálenost slov v dotazu. Uvažovat pouze sousednost slov nemusí být uspokojující. Mezi dvěma sousedními slovy v dotazu může být v textu například zájmeno, které význam fráze příliš neovlivňuje, ale znemožní zohlednění sousednosti slov, která ho obklopují.

Mějme dotaz „term1 term2 term3“. Pak uvažujme všechny dvojice slov *{a,b}*, tedy konkrétně *{term1, term2}, {term1, term3}, {term2, term3}*. Poté vypočítáme blízkost slov v dotazu $P_q(a,b)$, kde slova a,b tvoří dvojici:

$$P_q(a,b) = \min \left| pos_q(a) - pos_q(b) \right| \tag{2.15}$$

Dále určíme blízkost slov v dokumentu j ($a, b \in Q$):

$$P_j(a,b) = \min \left| pos_j(a) - pos_j(b) \right| \tag{2.16}$$

Vypočteme koeficient odchylky blízkosti dvojice slov v dotazu vůči blízkosti slov v dokumentu:

$$DQPC_j(a,b) = \begin{cases} \dfrac{k - \left|P_j(a,b) - P_q(a,b)\right|}{k}, pokud\ P_j(a,b) - P_q(a,b) \in (0,k\rangle\ \wedge a,b \in d_i \\ 0 \qquad\qquad\qquad\quad , v\ ostatních\ případech \end{cases} \tag{2.17}$$

kde *k* udává maximální vzdálenost dvojice slov.

Průměrný koeficient blízkosti dvojic slov v celém dotazu vůči dokumentu *j* vypočteme jako:

$$AverDQTP_j = \frac{\sum_{i=1}^{|N|} DQTP_j(N_{iA}, N_{iB})}{|N|} \tag{2.18}$$

kde *N* je množina dvojic (N_A, N_B) z dotazu.

Pro tento průměrný koeficient platí $AverDQTP_j \in \langle 0,1\rangle$, kdy „1“ znamená přesný výskyt fráze a „0“ znamená, že dokument neobsahuje frázi. Hodnoty mezi „0“ a „1“ značí různou míru podobnosti fráze uvnitř dokumentu.

Pro zjednodušení je možné tento algoritmus pro zjištění podobnosti výskytu fráze aplikovat až po vyhledání dokumentu podle čistého dotazu na dokumenty s ohodnocením vyšším než je stanovená mez. Toto ohodnocení poté modifikovat podle $AverDQTP_j$. Např.:

$$simP(d_j, q) = 0.7sim(d_j, q) + 0.3AverDQTP_j \tag{2.19}$$

Tento způsob zohlednění výskytu frází je jen teorií a jeho funkčnost na reálných datech zatím nebyla v praxi ověřena. Výpočet $AverDQTP_j$ bude pravděpodobně náročný, ale při vhodném předzpracování dat by mohl přinášet dobré výsledky.

3 Měření efektivity

Jedním z hlavních pojmů ve vyhledávání informací je *relevance* (důležitost, významnost). Relevance určuje, zda nalezené dokumenty obsahují pro uživatele informace, jaké svým dotazem žádá. Je to tedy poměrně subjektivní měřítko kvality závislé na kontextu informací. Systémy pro IR jsou hodnoceny podle relevance dokumentů k dotazu.

3.1 Testovací kolekce dokumentů

K tomu abychom mohli vzájemně porovnávat efektivnost různých IR modelů, musíme jejich efektivnost vyhledávání testovat pomocí stejných dat. K tomuto slouží testovací kolekce zaměřené na různé obory IR a obsahující různorodá data. Tyto kolekce obsahují jak samotné dokumenty, tak dotazy pro vyhledávání a samozřejmě seznam relevantních dokumentů pro každý jeden dotaz. Tato specifikace, které dokumenty jsou relevantní, je provedena odborníky na dané téma a považuje se za správnou. Takto tedy můžeme na dané kolekci dokumentů provádět vyhledávání pomocí definovaných dotazů a poté v porovnání s referenčními odpověďmi určit míru průniku množiny odpovědí s množinou referenčních odpovědí.

Zde je popis několika nejpoužívanějších kolekcí pro měření výkonnostních kvalifikátorů (zdroj [2]):

- *Reuters 21578* je kolekce 21578 novinových článků agentury Reuters z roku 1997. Články jsou za pomoci expertů rozděleny do mnoha kategorií.
- *RCV1* je další, mnohem větší, korpus agentury Reuters (Reuters Corpus volume 1). Tento obsahuje na 806791 anglicky psaných dokumentů.
- *Crangield kolekce* byla jedním z prvních korpusů umožňující precizní měření výkonnostních kvalifikátorů vyhledávání informací. Byla vytvořena v 50. letech 20 století a obsahuje 1398 abstraktů článků a množinu 225 dotazů s úplným určedním relevance všech párů dokument--dotaz.

Tvorbou těchto testovacích kolekcí se v současnosti zabývá například TREC (Text REtrieval Conference) a jejich data jsou k nalezení na volně na webu[1].

3.2 Úplnost a přesnost

Důležitými pojmy pro měření efektivity jsou úplnost (angl. recall) a přesnost (angl. precision). Uvažujme například dotaz specifikující touhu po informacích *I* a pro tyto informace množinu relevantních dokumentů R. Pomocí IR systému (který je hodnocen) získáme množinu odpovědí A. Pak je množina Ra průnikem množin R a A (jak je vidět na Obr. 3.1).

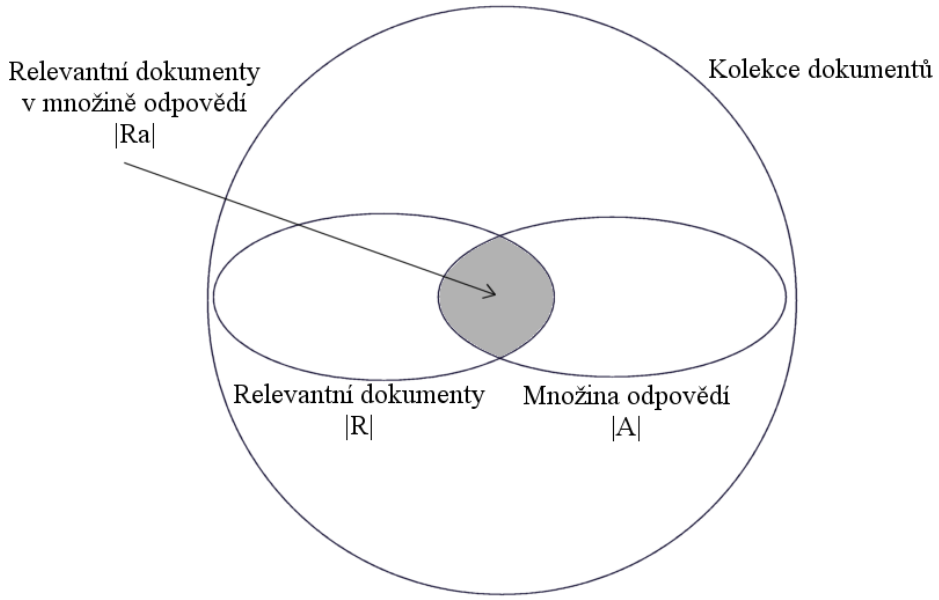
[1] http://trec.nist.gov/data.html

Úplnost vyjadřuje, jak velká část relevantních dokumentů byla nalezena [1], tedy schopnost nalézt všechny odpovídající dokumenty:

$$Recall = \frac{|Ra|}{|R|} \tag{3.1}$$

Přesnost vyjadřuje, jak velká část nalezených dokumentů je relevantní [1], neboli schopnost získat jen ty nejlépe ohodnocené dokumenty, které jsou obvykle ty odpovídající dotazu:

$$Precision = \frac{|Ra|}{|A|} \tag{3.2}$$



Obr. 3.1: Průnik množin relevantních dokumentů a množiny odpovědí [1]

Jak vysoká úplnost, tak i přesnost je důležitá pro IR systém. V ideálním případě by obě charakteristiky měly nabývat hodnoty 1, tj. tedy výsledek je složen ze všech a pouze relevantních dokumentů (viz. Obr. 3.2).

Úplnost a přesnost pro ohodnocené výsledky

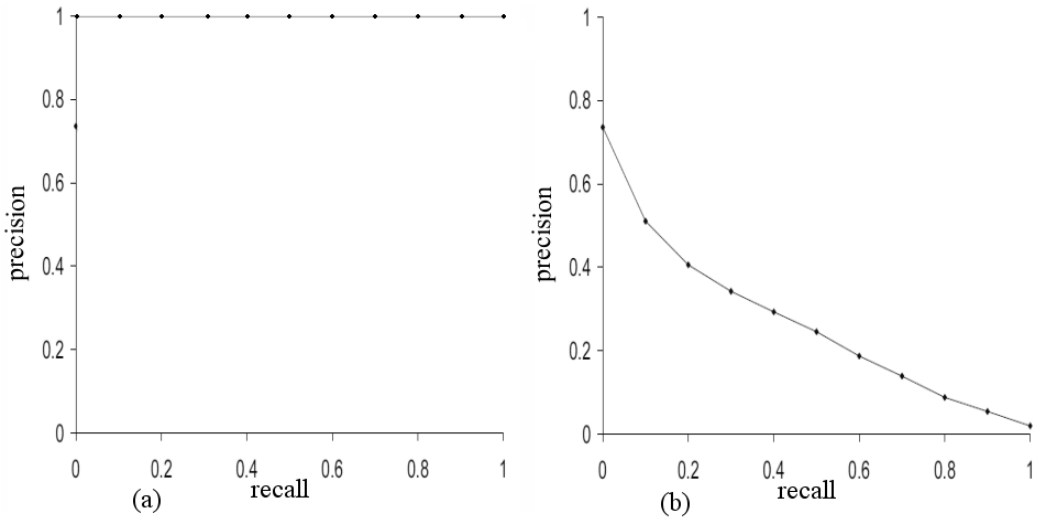
Metodou uvedenou výše můžeme měřit efektivitu pro množinu neohodnocených odpovědí (výsledek vyhledávání pomocí booleovského modelu). U množiny ohodnocených odpovědí nás však zajímá i přesnost hodnocení (ranking funkce), to znamená provést hodnocení s ohledem na uspořádání odpovědí. Toho dosáhneme výpočtem přesnosti pro různou úroveň úplnosti (např. postupně 10%, 20%,…,100%). Pak tedy uvažujeme v rovnici 3.2 prvních *x*% výsledků z množiny A (s vlivem na Ra). Případně můžeme přesnost a úplnost počítat pro každý relevantní výsledek, kdy množina odpovědí A je rovna odpovědím s ohodnocením stejným nebo vyšším, než má právě měřený výsledek. Abychom mohli tyto výsledky kombinovat s jinými, interpolují se hodnoty úplnosti do již zmíněných úrovní úplnosti (viz. Tabulka 3-1).

Úplnost	Přesnost
0	1
0,1	1
0,2	0,85
0,3	0,88
0,4	0,9
0,5	0,83
0,6	0,68
0,7	0,5
0,8	0,44
0,9	0,36
1	0,35

Tabulka 3-1 Přesnost v 11 úrovních úplnosti

Křivka přesnost-úplnost (častěji označována anglickým precision-recall) má obvykle zřetelný pilovitý průběh: pokud *(k+1)*-ní nalezený dokument je nerelevantní, pak je úplnost stejná jako pro prvních *k* dokumentů, ale přesnost klesá. V případě, že je relevantní, pak se zvýší přesnost i úplnost a křivka „jde vpravo a nahoru“. Tyto skoky se obvykle odstraňují a standardním způsobem, jak toho docílit, je výpočet interpolované přesnosti dle vzorce 3.3. Interpolovaná přesnost $Pinterp$ na určité úrovni úplnosti r se vypočítá jako nejvyšší nalezená přesnost na úrovních úplnosti $r' > r$.

$$Pinterp(r) = \max_{r' > r} p(r') \tag{3.3}$$



Obr. 3.2: (a) ideální a (b) obvyklé ohodnocení IR systému (dle [1])

Při hodnocení pro celou kolekci dotazů je možné spočítat průměrnou přesnost dle vzorce [1]:

$$AverP(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q} \tag{3.4}$$

Kde N_q je počet dotazů a $P_i(r)$ je přesnost pro dotaz i na úrovni úplnosti r.

MAP (Mean Average Precision)

Výše uvedené metriky nám dávají představu o přesném rozložení relevantních odpovědí v kolekci výsledků. Někdy je však příhodné použít jedno hodnotovou metriku, která bude představovat souhrn přesností nad celou množinou ohodnocených odpovědí. K tomu slouží například, v komunitě TREC nejpoužívanější[2], metrika *Mean Average Precision* (česky průměr střední přesnosti, dále jen MAP). Mezi používanými hodnotícími metrikami poskytuje MAP obzvláště dobrou rozlišovací schopnost a stálost.

Pro jednotlivou informační potřebu (jeden dotaz) je průměrná přesnost hodnota získaná pro kolekci prvních k dokumentů z množiny výsledků, po nalezení všech relevantních dokumentů. Tato hodnota je následně zprůměrována pro všechny dotazy. Formálně charakterizováno: Množina relevantních dokumentů pro konkrétní dotaz $q_j \in Q$ je $\{d_1, \dots, d_{m_j}\}$ a R_{jk} je množina prvních k ohodnocených výsledků, tedy těch do získání dokumentu d_k včetně. Pak se MAP vypočítá jako:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{3.5}$$

Když nejsou relevantní dokumenty nalezeny vůbec, přesnost v rovnici (3.5) nabývá hodnoty 0. Průměrná přesnost pro jeden dotaz aproximuje prostor pod precision-recall křivkou. MAP tedy značí průměrnou plochu pod touto křivkou pro celou množinu dotazů. S použitím MAP nejsou definovány žádné pevné úrovně úplnosti a není provedena interpolace přesnosti. Její hodnota pro testovací kolekci je tak aritmetickým průměrem přesností jednotlivých dotazů.

4 Zpracování textových dat

IR modely jsou vesměs univerzální, tedy nezávislé na datech, ve kterých mají umožňovat vyhledávání. Z toho důvodu je třeba provést vhodné zpracování textu, díky kterému získáme z dokumentu vhodná indexová slova. Toto zpracování je obvykle silně zaměřeno na konkrétní jazyk použitý v textu, protože charakteristiky slov i ostatních řetězců se mezi jazyky (skupinami jazyků) významně liší. Kvalita, se kterou jsou získána indexová slova, přímo ovlivňuje kvalitu chování celého IR systému.

4.1 Analýza textu

K získání indexových slov je třeba text dokumentu vhodně analyzovat a podle toho rozdělit na jednotlivé tokeny. Budeme uvažovat čistý text bez formátovacích značek nebo podobných struktur a zároveň se nebudeme zabývat získáním tohoto textu z konkrétních zdrojů.

Tomuto tématu se také věnuje [2], kapitola 2.2.

4.1.1 Tokenizace

První krok, který je třeba vykonat, je rozdělení textu na úseky znaků, které vnímáme jako slova, čísla, nebo jiné znaky (řetězce znaků), které mají daný význam, a odstranění nadbytečných znaků jako jsou interpunkční znaménka. Tyto úseky tvoří kandidáty na další zpracování a v konečném důsledku tvoří indexová slova (termy).

Nejsnazším způsobem, ale ne ideálním, by bylo nahradit všechny nepísmenné znaky bílými a pak podle bílých znaků (mezera, odřádkování, tabulátor, atp.) rozdělit text na tokeny. Tím bychom sice dospěli k získání vhodných tokenů, ale zároveň bychom tím ztratili mnohé jiné informace, které mohou uživatele zajímat. Dále jsou velkou komplikací čísla a jejich různorodý způsob vyjádření a použití. Zvažme, jak má být zpracováno datum, které může být zapsáno například jako „1.1.2010“. Má se rozdělit na 3 tokeny „1“, „1“ a „2010“ podle tečky, má být zachováno v původním tvaru nebo tečku vyjmout a vytvořit celistvý „112010“. Z pohledu vyhledávání by asi bylo ideální datum rozpoznat v několika tvarech a převést na jednotný tvar. Ale toto je jen jeden případ z mnoha, u kterých už nemusí být jasná odpověď.

Univerzální řešení, které by neznevýhodňovalo určité typy řetězců, asi neexistuje. Vždy by se našly takové případy (např. zdrojové kódy), že by je proces tokenizace nezpracoval korektně. Je třeba se rozhodnout mezi řešením takovým, které se bude snažit poskytnout dobrý podklad pro vyhledávání (tj. co nejobecnější tvar řetězce), a tím, jenž uchová původní tvar řetězce.

Během zpracování textu se tokenizér potýká i s vlastnostmi jako je velikost písmen a v případě českého jazyka i diakritika.

Velikost písmen

Zdálo by se, že převod všech písmen na malé, je účinné řešení. Ale pokud uvážíme dvojici slov *FIT* a *fit*, zájemce o studium informatiky asi nebude rád, když mu vyhledávací systém vrátí dokumenty o

cvičení a zdravé výživě. Je tedy na rozvaze jak se má systém chovat, zda obecně a konvertovat všechna velká písmena na malá, nebo zda má zachovávat sémantiku skrytou ve velikosti písmen a měnit velikost písmen například jen u slov na začátku věty.

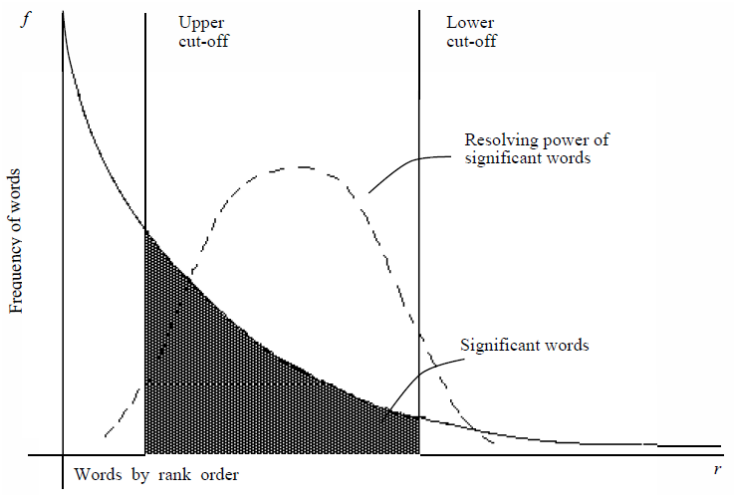
Diakritika

Jedná se o podobný případ jako u velikosti písmen. Pro „univerzální“ řešení můžeme odstranit všechnu diakritiku, což je, vezmeme-li v úvahu, kolik uživatelů píše bez použití diakritiky, řešení, které má dobrý vliv na množství získaných výsledků. Potom se ale sémanticky rozdílná slova jako jsou například *koš* a *kos* stanou nerozlišitelná. Opět je tedy na rozhodnutí, zda má systém poskytovat více odpovědí, ale s rizikem, že nebudou zcela relevantní, nebo má uživatel zadávat svůj dotaz korektně k jeho sémantice.

4.1.2 Eliminace častých slov

Častá slova, jak jsme si již zmínili v kapitole 2.3, mají při výpočtu *idf* velice nízkou, spíše až žádnou vypovídací hodnotu o textu, v němž se vyskytují. V tomto ohledu by nám tato slova ani moc nevadila, ale například při výpočtu normované frekvence slov (rovnice 2.3) může v jednom dokumentu enormně se vyskytující slovo zbytečně znevýhodnit tento dokument od ostatních. Další nevýhodou častých slov je prostor, který zabírají v indexu (obzvláště pak v pozičním indexu slov v dokumentech).

Řešením, jak se vypořádat s negativy spojenými s častými slovy, je tato slova z kandidátů na indexová slova odstranit. K tomu potřebujeme seznam těchto častých slov (obvykle se nazývají stop slova – stopwords), který můžeme buď získat předem vytvořený (například na webu University of Neuchâtel[2]) nebo sami vytvořit z dostatečně velké báze dokumentů. Slova s nízkou *idf* (dle [4] < 0.2) jsou vhodnými kandidáty na seznam stop slov. Dnešní trend však vede k používání malého seznamu stop slov (desítky místo stovek), který obsahuje převážně spojky atp. [2].



Obr. 4.1: Charakteristika význačných slov [5]

[2] http://members.unine.ch/jacques.savoy/clef/index.html

4.1.3 Lemmatizace a Stemming

Slova v dokumentech mají z gramatických důvodů různé tvary, kdy slova s různými koncovkami, mají víceméně stejný význam (například student, studentka, studenti). Abychom mohli nad těmito tvarově i významově podobnými slovy efektivně vyhledávat, je potřeba je převést na společný základní tvar. Tímto docílíme v konečném důsledku i menšího slovníku IR systému, takže jeho prostorové nároky budou taktéž menší.

K získání základního tvaru slova je možné využít lemmatizace, kdy se využívá slovníkové a morfologické analýzy slov. Tento proces dosahuje dobrých výsledků, avšak pouze v případě, že daný tvar je obsažen ve slovníku.

Další možností, jak se dopracovat k základnímu tvaru slova, je proces zvaný stemming. Ten pomocí primitivní heuristiky analyzuje slovo a odřízne jeho příponu, případně ještě doplní do pravděpodobně korektního tvaru. Výstup stemmingu nemusí plně odpovídat gramaticky správnému základnímu tvaru slova (kořene). Spokojíme se s tím, že pravděpodobně významově stejná a tvarově podobná slova převede do jednotné formy. Nejznámější je Porterův stemmer (ukázka viz Obr. 4.2) navržený pro angličtinu, jehož princip se převzal i pro jiné jazyky.

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Obr. 4.2: Ukázka principu fungování Porterova stemmeru [2]

Porterův stemmer funguje na principu pravidel, která definují znaky na konci slova, které mají být nahrazeny jinými nebo úplně odstraněny. Do toho vstupuje ještě vliv délky slova na výběr možných pravidel k použití. Při použití dostatečně velké množiny pravidel by měla být snížena šance na oříznutí 2 významově různých slov na stejné. I přesto však toto automatické oříznutí slova na kořen není dokonalé a nedosahuje pro běžná slova takových výsledků jako při použití slovníkových stemmerů.

Stejně jako v předchozích krocích zpracování textu, i použití převodu na základní tvar slova má různý dopad na efektivitu vyhledávání. Pro některé dotazy může být výhodná generalizace slov, pro jiné může způsobit snížení přesnosti (viz kapitola 3.2).

4.2 Využití sémantických slovníků

Dalším způsobem, jak se můžeme pokusit zvýšit efektivitu vyhledávání je použití sémantických slovníků, konkrétně za účelem získání synonym slova. Jejich použití ve fázi zpracování dokumentů (indexace) nepřichází v úvahu. Informace o dokumentech můžeme zobecňovat, nikoliv však rozšiřovat. Správným krokem však může být rozšíření dotazu o synonyma jednotlivých slov.

Přímé rozšíření dotazu

Slovník synonym (neboli thesaurus) obsahují dvojice *slovo – jeho synonymum*. To nám dává možnost získat pro každé slovo seznam jeho synonym. Můžeme pak rozšířit dotaz o tato synonyma. Z pohledu zachování tvaru dotazu, by bylo vhodné místo pouhého přidání synonym, nahradit původní slovo množinou synonym tohoto slova (včetně něj). Například:

{term_1, term_2} => {{term_1, synonymum_termu_1,...},{term_2, synonymum_termu_2,... }

I přesto, že vektorový model vyhledávání nebere ohled na pořadí nebo sousednost slov v dokumentu / dotazu, je vhodné tuto strukturu dotazu ponechat pro další využití a až v konečné fázi zpracování převést dotaz na vektor.

Agregace termů synonym slova

Pouhé rozšíření dotazu o synonyma slova může vést ke snížení ohodnocení dokumentů, v nichž se tato synonyma nevyskytují. Obráceným přístupem je přepočítání vah termů pro každý dokument způsobem, kdy se dotazovaný term (tedy term obsažený v dotazu) a všechny termy jeho synonym budou považovat za jeden term a tím bude onen dotazovaný term.

Rozšíření dotazu o synonyma může vést k lepším výsledkům vyhledávání, ale zároveň může uživateli poskytovat výsledky na pro něj nevyhovující výrazy. Proto by mělo být použití rozšíření dotazu o synonyma volbou, kterou lze zakázat.

5 Specifikace a návrh systému

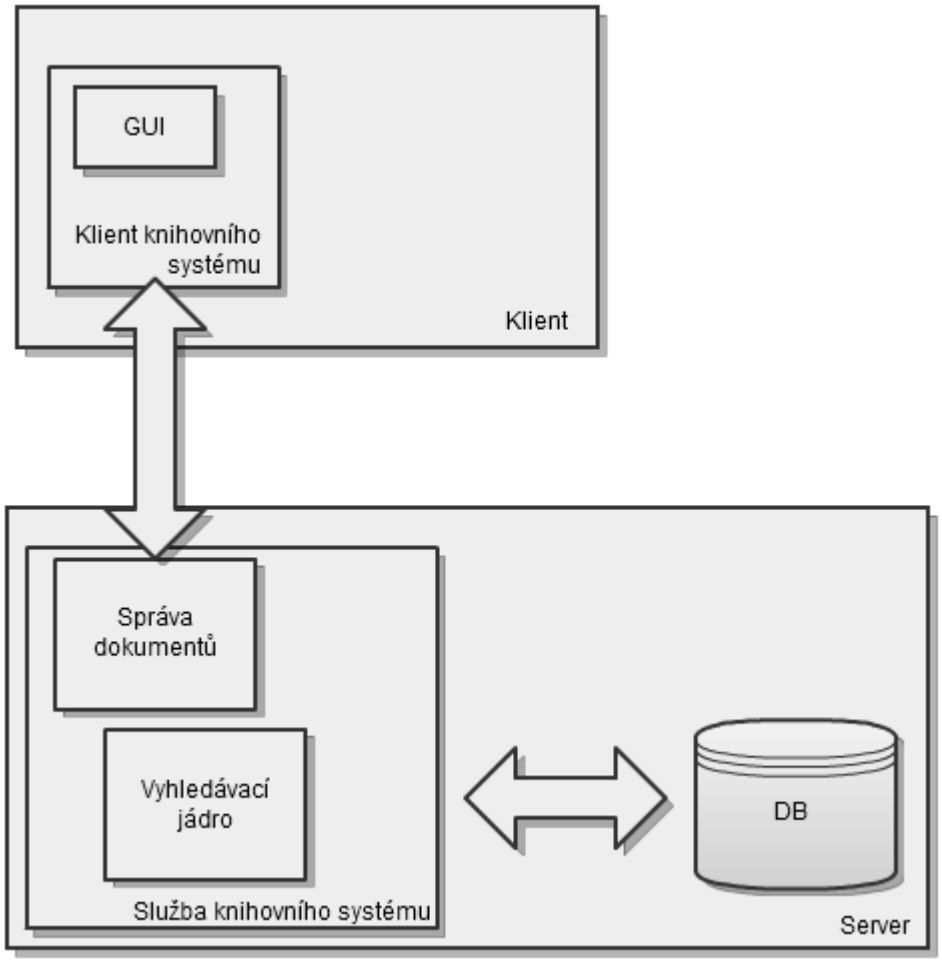
V této kapitole nejprve rozebereme návrh systému pro zpracování textových dokumentů z hlediska vyhledávání v nich, jejich uchování a vyhledávání nad nimi. Dále se seznámíme s návrhem demonstrační aplikace, která bude využívat prvně jmenovaný systém pro vyhledávání diplomových prací dostupných na webu Fakulty Informačních Technologií VUT v Brně. Systém bude pro ukládání dat využívat běžnou relační databázi.

5.1 Neformální specifikace

Cílem práce je vytvořit aplikaci, která bude umožňovat správu dokumentů a vyhledávání informací v jejich obsahu. Konkrétně se jedná o PDF dokumenty absolventských prací a metadata pro jejich identifikaci a stručný popis. Samotné vyhledávání se bude dít nad texty z PDF dokumentů. Systém bude pro vyhledávání využívat systém popsaný v kapitole 2. Konkrétně vektorový model pro vyhledávání informací, rozšíření dotazu o synonyma v něm obsažených slov, s pokud možno co nejnižším negativním dopadem na efektivitu systému, a modifikaci ohodnocení relevantnosti dokumentů vůči dotazu metodou výpočtu podobnosti výskytu fráze z dotazu v dokumentu, popsanou v podkapitole 2.4.2. Samotná aplikace by měla poskytovat přívětivé uživatelské rozhranní a umožňovat víceuživatelský přístup k vyhledávání informací.

5.2 Architektura systému

Systém vyhledávání informací bude rozdělen na 2 logické celky: vyhledávací jádro a demonstrační aplikaci. Samotnou demonstrační aplikaci bude tvořit služba běžící na serveru, se kterou bude komunikovat klient zprostředkovávající uživatelské rozhraní (Obr 5-1). Vyhledávací jádro se bude starat o logiku vyhledávání informací, zatímco služba bude schraňovat metainformace o dokumentech, ve kterých se vyhledává, a volat funkce jádra pro indexování dokumentů a vyhledávání informací v nich.



Obr 5-1 Navržená architektura systému pro vyhledávání v elektronických knihovnách

5.3 Specifikace a návrh vyhledávacího jádra

Navrhovaný systém vyhledávání informací v dokumentech vychází primárně z principu vektorového modelu popsaného v kapitole 2. Dále bude používat rozšíření dotazu o synonyma v něm obsažených slov a ohodnocení výsledků hledání bude upraveno dle podobnosti výskytu fráze z dotazu v jednotlivých dokumentech. Tento princip použití pozičních indexů byl popsán v kapitole 5.2.

5.3.1 Uchování dokumentu

Prvně si je třeba si ujasnit jaké informace o dokumentu a v jaké formě budeme ukládat. Z principu fungování vektorového modelu pro vyhledávání informací víme, že potřebujeme pro každý dokument znát jeho vektor vah. Váhy jsou vypočteny dle tf–idf schéma. Výpočet TF je ovlivněn jen jednotlivými dokumenty, tedy jeho hodnota je v čase při neměnnosti dokumentu stálá. Oproti tomu IDF termu se s měnícím se počtem dokumentů také mění.

Základním prvkem je tedy *term*, neboli indexové slovo, a *dokument*, který je složený z termů. Z důvodu vyhledávání podobnosti fráze v dokumentu je potřeba uchovávat nejen informaci o výskytu

termu v dokumentu, ale i o pozicích, na kterých se v něm vyskytují. Definujme tedy entitní třídy pro reprezentaci těchto informací a vztahy mezi nimi. Každá entita bude obsahovat jednoznačný celočíselný identifikátor. Pro důraz na použití entit v systému vyhledávání informací (information retrieval) použijeme v jejich jménech prefix *Ir*.

IrTerm je definován svojí hodnotou (*Value*), tedy textovou reprezentací slova, které vyjadřuje. Dále má každý term svoji dokumentovou frekvenci *DF* a z ní vyplývající *IDF*. Zdálo by se, že je zbytečné uchovávat zároveň hodnoty DF a IDF, ale z důvodu rychlejšího přepočtu IDF je dobré mít dokumentovou frekvenci termu uchovanou, než znovu a znovu hledat a počítat dokumenty, v nichž se term vyskytuje.

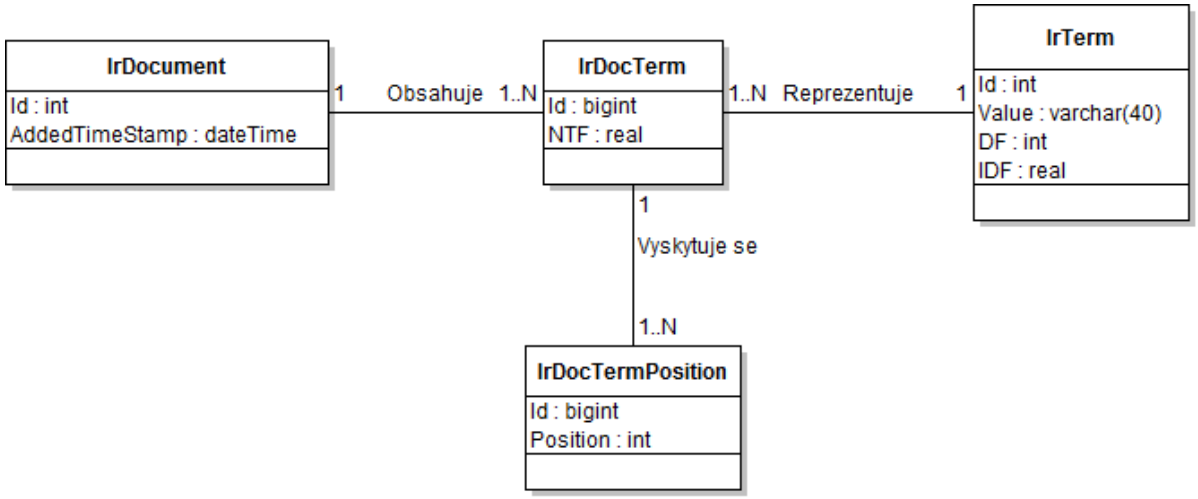
IrDocument reprezentuje naindexovaný dokument, který obsahuje některé z termů. U něj nás může zajímat čas, kdy byl do systému vyhledávání přidán (AddedTimeStamp).

IrDocTerm – touto entitou je reprezentován vztah mezi dokumentem a termem. U termu v dokumentu nás zajímá jeho normovaná frekvence (*NTF*).

IrDocTermPosition – pozici výskytu termu v dokumentu definuje entita Ir*DocTermPosition* a její atribut *Position*.

Vztahy mezi jednotlivými entitami jsou definovány následovně (viz Obr 5-2):

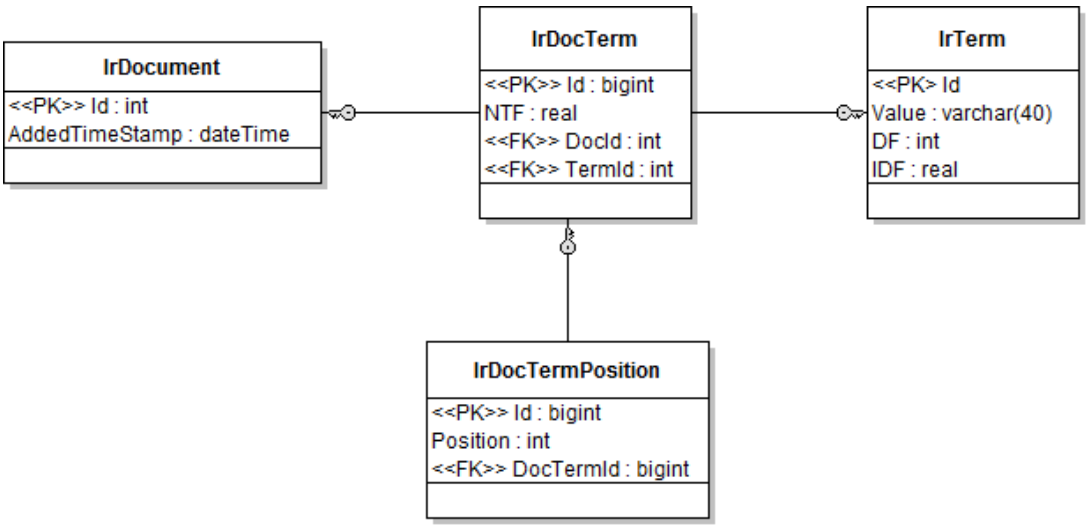
- Dokument obsahuje jeden a více výskytů termu v dokumentu (IrDocTermu), který je zároveň unikátní pro každý dokument.
- IrDocTerm reprezentuje jeden konkrétní term a tento term může být reprezentován větším množstvím IrDocTerm-ů.
- IrDocTerm se vyskytuje na jednom a více místech v dokumentu (IrDocTermPosition) a tento výskyt je pro daný dokument unikátní.



Obr 5-2 ER diagram systému vyhledávání

Databázové schéma

Protože k uložení dokumentu bude sloužit relační databáze, je třeba převést ER diagram na databázové schéma. Vyhodnotíme vztahy mezi entitami a dle toho založíme primární a cizí klíče. Jako primární klíče slouží vždy celočíselné Id a to z důvodu pozdějšího použití ORM pro práci s databází. Databázové schéma pro jádro systému vyhledávání můžeme vidět na Obr 5-3.



Obr 5-3 Diagram znázorňující databázové schéma

5.3.2 Zpracování textu

Aby bylo možné dokument uložit, musí se jeho obsah, tedy text, náležitě zpracovat. Obecné postupy, jak toho docílit, byly rozebrány v kapitole „Analýza textu“. Cílem tohoto procesu je z prostého textu získat kolekci vyhovujících a unikátních termů. Kroky zpracování textu jsou následující:

- rozdělení textu na tokeny
- normalizace velikosti písmen
- eliminace stop slov
- odstranění diakritiky
- získání základního tvaru (stemu) z tokenu
- agregace dle hodnoty stemu – získání počtu a pozic výskytů v textu

V případě zpracovávání textu z dotazu je třeba, pokud je to vyžadováno, navíc získat synonyma pro každý stem, respektive token, ze kterého byl vytvořen. Po výše uvedených krocích tedy provedeme:

- získání kolekce synonym pro každý stem
- odstranění diakritiky synonym
- získání základního tvaru (stemu) ze synonyma

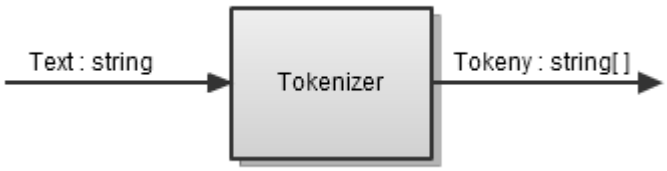
Tokenizer

K rozdělení textu na tokeny poslouží jednoduchý tokenizer, který bude mít za úkol rozdělit text na podřetězce dle definovaných symbolů - oddělovačů. Mezi základní oddělovače patří tzv. bílé znaky a interpunkční znaménka. Po zevrubné analýze běžných textů si však definujeme ještě další typy symbolů sloužících jako oddělovače.

Je důležité definovat co největší počet symbolů, které nejsou od slov odděleny výše uvedenými separátory a které pak znehodnocují slovo, jehož se tak stávají součástí. Prvním příkladem jsou různé závorky, které z vnitřní strany nemají mezeru. Pak existují různé typy uvozovek a apostrofů, které se používají například při citaci. Dále můžeme za nežádoucí považovat spojení tokenů matematickými symboly, z nichž vybereme alespoň ty základní.

Získáme tedy následující oddělovače tokenů:

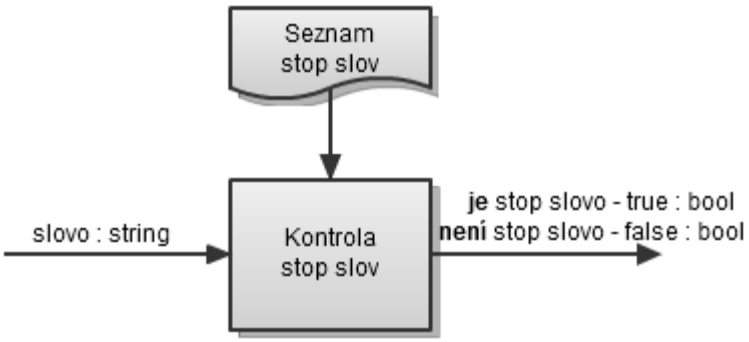
- bílé znaky (mezera, nezlomitelná mezera, tabulátor, odřádkování, …)
- interpunkční znaménka (tečka, čárka, otazník, vykřičník, dvojtečka, středník, …)
- závorky (kulaté, hranaté, …)
- různé typy uvozovek
- matematické symboly (+, -, /, *)



Obr 5-4 Funkce tokenizéru

Eliminace stop slov

K odstranění častých slov v textech využijeme jednoduchou kontrolu spočívající ve využití předem daného, tedy statického, seznamu stop slov. Tento soubor bude načítán z externího zdroje pro možnost jeho předefinování. Kontrola slova bude spočívat ve vyhledání předaného slova v kolekci stop slov. V případě nalezení bude označeno za stop slovo a naopak.



Obr 5-5 Kontrola stop slov

Soubor s kolekcí stop slov lze získat z webových stránek University of Neuchâtel[3].

Stemming slov

K získání základního tvaru slova můžeme využít buďto slovníků, které naleznou tento tvar, pokud ho obsahují, nebo algoritmů pro stemming slov. Pro možnost porovnání vlivu Stemmeru na vyhledávání bude použito více typů stemmerů. Všechny tak musí využívat stejné rozhraní, aby je bylo možné zaměnit.

Slovníkový stemmer

Jako slovníkový stemmer je možné využít knihoven pro práci se slovníky ispell a jeho variací, například hunspell nebo myspell, který je využívaný balíkem OpenOffice, jenž je využívá ve svém modulu pro automatickou kontrolu textu. Díky tomu jsou dostupné slovníky v mnoha jazykových variacích.

Ukázka souboru pro stemming ve formátu hunspell

Definice základních tvarů slov:

```
influence/Z
influenza/ZQ
infolinka/ZQ
informace/Z
informatický/YRN
```

Definice přípon:

```
SFX Z   e           i         [^g]e
SFX Z   e           í         [^g]e
SFX Z   e           ím        [^g]e
SFX Z   e           ích       [^g]e
SFX Z   0           mi        [^g]e
```

Příklad: Ze slova *informace* lze změnou přípon získat tvary *informaci, informací, informacím, informacích a informacemi*. Respektive opačným postupem získat základní tvar slova, tedy *informace*.

---

[3] *IR Multilingual Resources at UniNE* <http://members.unine.ch/jacques.savoy/clef/index.html>
Tyto stránky obsahují užitečné informace pro zpracování textu pro potřeby systémů vyhledávání informací. Jsou zde dostupné stemmery a seznamy stop slov pro převážně evropské jazyky, mezi nimiž nechybí ani čeština. Dále sdružuje odkazy na práce popisující jednotlivé jazyky a jejich gramatiky

Analytický stemmer

Ve variantě analytického stemmeru (viz. kapitola 4.1.3), který ořezává slovo na základě jeho zevrubné analýzy, se pro český jazyk nabízejí 2 varianty z webu University of Neuchâtel[3]. První je lehká verze stemmeru, která se snaží odstranit koncovky podstatných jmen, přídavných jmen, přivlastňovacích zájmen a odstraňuje měkké konce slov. Druhá varianta se prezentuje jako agresivní s tím, že oproti lehkému stemmeru odstraňuje konce u zdrobnělin, augmentativní přípony, komparativy a přípony odvozenin od podstatných jmen. Obě varianty jsou napsané v jazyce Java a dostupné volně k použití pod BSD licencí. Dle komentáře k nim by mělo být výhodnější použití lehké verze. V testovací aplikaci však budou využity obě verze pro možnost porovnání jejjich vlivu na efektivitu vyhledávání.

Získání synonym slova

Pro získání synonym slova, bude využito již výše zmíněných slovníků hunspell, k nimž lze přistupovat pomocí knihoven dostupných v mnoha jazycích. K získání synonym pak stačí zavolat příslušnou metodu, která v případě nálezu vrátí kolekci synonym požadovaného slova.

Jen pro představu si ukážeme, jak vypadá obsah slovníku synonym v případě hunspell, konkrétně jeden záznam slova a výčet jeho synonym.

Ukázka obsahu souboru slovníku synonym ve formátu pro hunspell:

```
vyhladit|1
-|zahladit|vyrovnat|vykořenit|vymazat|zničit|zabít|vyhubit
vyhladovělý|1
-|hladový
vyhlazení|1
-|vykořenění|vyhubení
vyhledat|1
-|vypátrat
```

Pro korektní získání synonyma je potřeba nejprve slovo převést na základní tvar, čehož lze docílit taktéž použitím knihoven pro práci se slovníky hunspell.

5.3.3 Indexace dokumentů

Proces indexace dokumentu slouží k převedení řetězcové reprezentace dokumentu do formy vhodné pro vyhledávání a zároveň její perzistenci pro možnosti pozdějšího použití, kterou jsem navrhl v kapitole „Uchování dokumentu“.

Kroky potřebné pro indexaci dokumentu:

1. Získání termů, jejich NTF a pozic
2. Získání již existujících termů z db
3. Vytvoření nových termů
4. Vytvoření Dokumentu,

5. Vytvoření DocTermu a DocTermPosition z hodnot získaných v prvním kroku
6. Inkrementace DF termů
7. Uložení všech nových a upravených objektů
8. Přepočet IDF všech termů

Odstranění dokumentu

V případě odstraňování dokumentu ze systému pro vyhledávání je potřeba zajistit, aby se provedlo konzistentně odebrání všech jeho částí, tedy *DocTerm*-ů, jejich *DocTermPosition* a samotného *Document*. Zároveň se musí provést dekrementace hodnoty DF u *Term*-ů, které se vyskytovaly v dokumentu, případně jejich odstranění, pokud je DF rovno nule. Nakonec je třeba provést přepočet hodnoty IDF u všech *Term*-ů.

5.3.4 Vyhledávání dokumentů

Proces vyhledávání zahrnuje v prvé řadě zpracování dotazu a jeho převedení do formy nutné k použití samotného vyhledávacího algoritmu. Jak již bylo zmíněno, vyhledávání bude založeno primárně na vektorovém modelu pro vyhledávání informací s využitím rozšíření dotazu o synonyma a modifikace ohodnocení dokumentu dle podobnosti výskytu fráze z dotazu v dokumentu.

Zadaný dotaz je nejdříve potřeba zpracovat na termy, stejně jako při indexaci dokumentu. Nyní se navíc pro pozdější použití naleznou pro každý z termů jeho synonyma. Poté vyhledat dokumenty obsahující alespoň některý z termů (nebo termů synonym) dotazu a pro tyto dokumenty získat NTF, IDF a pozice termů v nich obsažených. Získané termy v dokumentech je poté možné agregovat dle hodnoty termu v dotazu a jeho synonym. Z těchto dat se pak vytvoří váhové vektory a provede se výpočet podobnosti dle vektorového modelu (kapitola 2.3). Poté je možné provést korekci ohodnocení podobnosti výpočtem podobnosti výskytu fráze (kapitola 2.4.2) využitím známých pozic termů v dokumentu.

Kroky vykonávané při vyhledávání dokumentů:

1. Získání termů, jejich NTF a pozic z dotazu. V případě potřeby nalezení jejich synonym
2. Získání IDF pro termy dotazu z databáze.
3. Získání dokumentů obsahujících alespoň některé termy (případně i jejich synonyma) z dotazu, NTF a IDF pro všechny jejich termy. V případě potřeby (zjištění podobnosti výskytu fráze) i kolekci pozic pro termy (a jejich synonyma) vyskytující se v dotazu
4. Vytvoření vektoru vah pro dotaz
5. Vytvoření vektoru vah pro každý dokument
   5.1. Pokud je použita expanze dotazu o synonyma, provede se agregace synonym a jim odpovídajícího termu dotazu do tohoto termu. Zároveň s tím se upraví hodnota NTF přepočtem a IDF.
6. Pro každý dokument – expanze vektoru vah dotazu o dimenze o nulové velikosti pro váhy termů obsažených v dokumentu, ale ne v dotazu
7. Výpočet míry podobnosti dokumentu a dotazu využitím Vektorového modelu pro vyhledávání informací.

8. Výpočet míry podobnosti výskytu fráze z dotazu v dokumentu.
9. Kombinace míry podobnosti dokumentu a dotazu s mírou podobnosti výskytu fráze.
10. Vrácení všech dokumentů s ohodnocením vyšším než je minimální stanovená mez.

5.4 Specifikace a návrh demonstrační aplikace

Demonstrační aplikace, jak již bylo zmíněno v úvodu této kapitoly, má umožňovat vyhledávání informací v diplomových pracích, jež jsou dostupné na webu FIT VUT v Brně. Ty jsou zde uloženy ve formátu PDF. Cílem je tedy vytvořit aplikaci, která bude uchovávat a spravovat informace o studentských pracích a k vyhledávání bude využívat navržené vyhledávací jádro.

Tato aplikace bude navržena s využitím konceptu architektury zaměřené na služby neboli SOA (Service Oriented Architecture). Vznikne tak služba, konzolová aplikace, která bude vykonávat celou business logiku zahrnující veškerou práci s dokumenty, a klienta, aplikaci s grafickým uživatelským rozhraním, který bude pouze volat funkce (metody) poskytované rozhranním služby. Tento přístup je výhodný z důvodu rychlosti přístupu do databáze, kdy služba běží například na stejném (nebo z pohledu rychlosti spojení blízkém) počítači jako databázový systém a klient komunikuje přes linku s vyšší latencí. Zároveň s databází komunikuje pouze jeden program, který může využívat určitého stupně cachování a zajišťovat konzistenci dat.

Specifikace vybraných případů užití

Případ použití: Editace metadat dokumentu
Identifikátor: UC1
Popis: Správce vyhledá konkrétní dokument a upraví jeho meta informace.
Předpoklady Správce je v sekci „Správa dokumentů“.
Aktéři Správce
Hlavní tok 1. Správce zadá název dokumentu a stiskne tlačítko vyhledat. 1.1. Systém zobrazí kolekci odpovídajících dokumentů 1.2. Systém nenalezl odpovídající dokumenty a hlavní tok končí. 2. Správce vybere konkrétní dokument. 3. Správce upraví meta informace vybraného dokumentu a klikne na tlačítko uložit. 4. Systém provede uložení změn meta informací dokumentu do databáze.
Výjimky Neplatné spojení se serverem Neplatně hodnoty meta informací
Výjimka: Neplatné spojení se serverem
Identifikátor: UC1.E1
Vedlejší tok 1. Případ použití začíná po kroku 1 nebo 3 hlavního toku. 2. Systém zobrazí chybovou hlášku o nemožnosti spojení se serverem.

Výjimka: Neplatně hodnoty meta informací
Identifikátor: UC1.E2
Vedlejší tok 1. Případ použití začíná po kroku 3 hlavního toku. 2. Systém zobrazí chybovou hlášku o duplicitní hodnotě „Název dokumentu“

Případ použití: Vyhledání informace službou
Identifikátor: UC2
Popis: Služba provede požadavek na vyhledání a vrátí ohodnocené dokumenty
Předpoklady Služba může komunikovat s databází
Aktéři Knihovní vyhledávací služba (dále jen služba)
Hlavní tok 1. Hlavní tok začíná v okamžiku, kdy služba obdrží požadavek na vyhledávání. 2. Služba použije funkci vyhledávacího jádra pro získání kolekce dle relevance ohodnocených id dokumentů. 2.1. Pokud je kolekce prázdná, služba vrátí prázdnou kolekci výsledků. 3. Pro všechna id dokumentů se získá název a popis. 4. Služba vrátí kolekci výsledků hledání složených z ohodnocení, id, názvu a popisu dokumentu.
Výjimky Nejsou.

Případ použití: Vyhledání informace
Identifikátor: UC3
Popis: Uživatel vyhledá informaci a získá pdf dokument.
Předpoklady Uživatel je v sekci „Vyhledávání informací“.
Aktéři Běžný uživatel (dále jen uživatel)
Hlavní tok 1. Uživatel zadá dotaz a nastaví parametry vyhledávání. Poté stiskne tlačítko vyhledat. 2. Systém vyhledá dokumenty obsahující dotazovanou informaci 3. Systém uživateli se zobrazí kolekce dokumentů seřazená sestupně dle míry jejich relevance vůči dotazu. 4. Uživatel si dle popisu dokumentů vybere konkrétní dokument. 5. Stisknutím tlačítka „stáhnout PDF“ u vybraného dokumentu si uživatel uloží dokument na disk.
Výjimky Neplatné spojení se serverem
Výjimka: Neplatné spojení se serverem
Identifikátor: UC3.E1
Vedlejší tok 1. Případ použití začíná po kroku 1 hlavního toku. 2. Systém zobrazí chybovou hlášku o nemožnosti spojení se serverem.

5.4.1 Knihovní vyhledávací služba

Jak bylo výše zmíněno, služba bude využívat navrženého vyhledávacího jádra. To jí umožní jednoduše přidávat (indexovat), odebírat a vyhledávat dokumenty.

Uchování informací o knihovním dokumentu

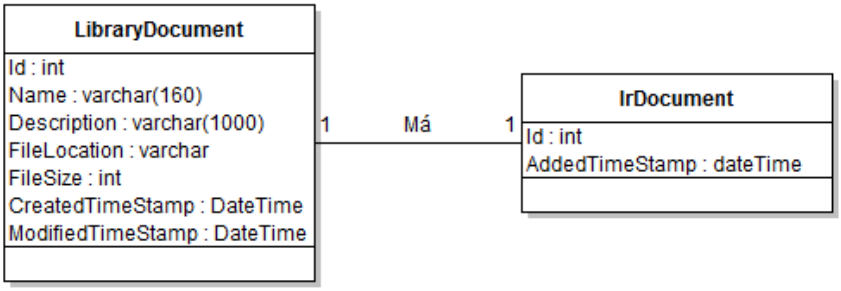
Vzhledem k nulovým informacím, které o sobě poskytuje entita dokument ve vyhledávacím jádru, pracuje služba pouze s dokumenty, které popisuje jen jejich *Id* (viz. entita IrDocument Obr 5-2). Je potřeba tedy definovat entitu *LibraryDocument* obsahující potřebné informace pro systém digitální knihovny. Mezi ty, pokud uvažujeme data o studentských pracích dostupná na webu (Obr 5-7), patří:

- *Název* dokumentu – skládající se ze jména studenta a názvu jeho práce.
- *Popis* dokumentu – který tvoří abstrakt práce
- PDF soubor s prací – samotný soubor budeme z důvodu velikosti uchovávat na disku. Zajímá nás tedy *cesta k souboru* a pro potřeby jeho automatické aktualizace (reindexování pro vyhledávání) se spokojíme s porovnáním jeho *velikosti*.

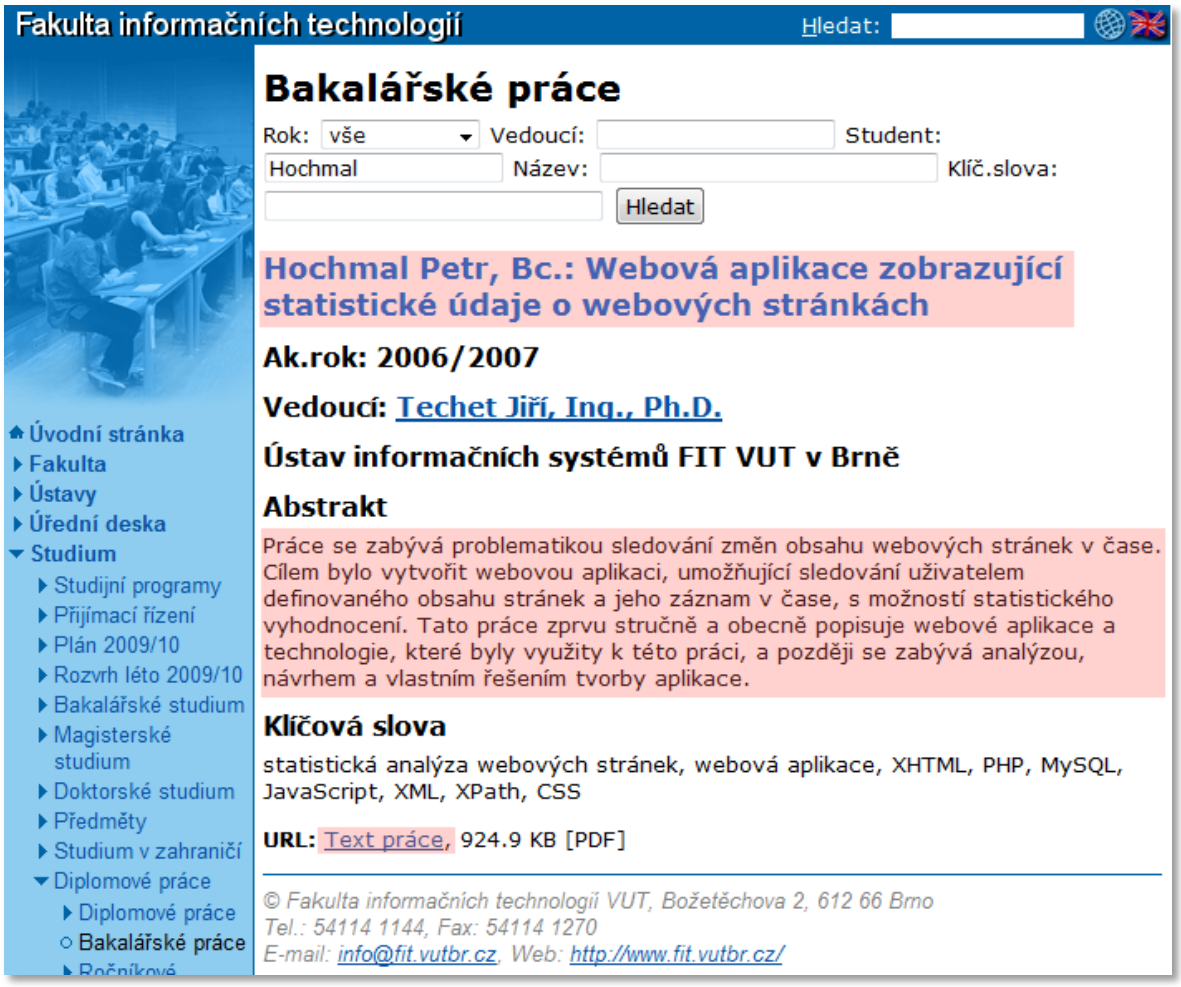
K těmto informacím je možno ještě přidat:

- *čas vytvoření* knihovního dokumentu
- *čas aktualizace*

Definujme tedy entitu *LibraryDocument*, která je ve vztahu 1:1 s entitou *IrDocument*, jak můžeme vidět na Obr 5-6

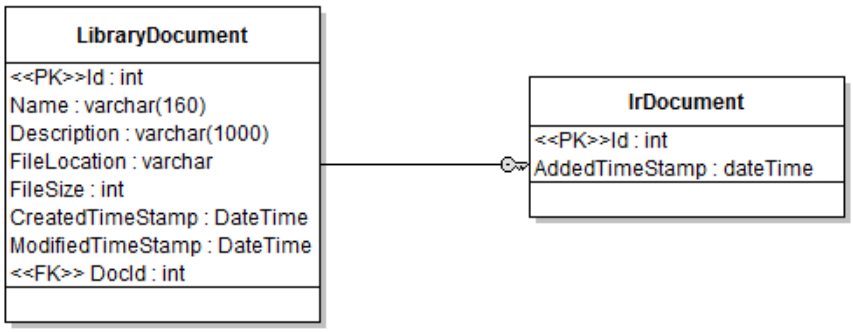


Obr 5-6 EntitaLibraryDocument a její vazba na entitu IrDocument systému vyhledávání



Obr 5-7 Část obsahu webové stránky s detailem bakalářské práce. Požadované informace jsou zvýrazněny.

Schéma databázové tabulky (Obr 5-8) z ní vytvoříme definováním primárního klíče na sloupci *Id* a přidáním sloupce *DocId* s cizím klíčem do tabulky *IrDocument*.



Obr 5-8Schéma tabulky LibraryDocument a vztahu k IrDocument

Služby poskytované klientům

Pro popis samotné služby musíme stanovit komunikační rozhraní, které definuje zprávy (metody) poskytnuté klientům, a pomocí kterého bude samotná komunikace mezi klientem a službou probíhat. Základní činností klienta má být správa dokumentů, musíme tedy poskytnout možnost vyhledání dokumentů dle jména, uložení editovaného a nového dokumentu a také jeho smazání. Hlavní náplní bude vyhledávání dokumentů, tedy získání dokumentů ohodnocených dle toho, jak odpovídají dotazu. Z těchto výsledků budeme chtít získat původní zdroj, tedy práci ve formátu PDF. Aplikace bude navíc obsahovat robota (angl. crawler) pro automatické získávání diplomových prací (popsáno v sekci „Automatické zpracování dokumentů“) a tak musí být umožněno spuštění, zastavení a zjištění aktuálního stavu tohoto robota.

Služba tedy bude poskytovat rozhraní pro:

- *Vyhledání dokumentu* – podle předaného dotazu a parametrů hledání (rozšíření o synonyma, podobnost fráze) vrátí kolekci ohodnocených LibraryDokumentů.
- *Získání PDF* – pro vybraný LibraryDokument vrátí jeho PDF soubor.
- *Vyhledání dokumentů pro editaci* – vyhledání předaného řetězce v názvu LibraryDokumentů a vrácení kolekce odpovídajících.
- *Vytvoření nového dokumentu* – předán název, popis a pdf soubor
- *Smazání dokumentu* – dle Id LibraryDokumentu
- *Editace dokumentu* – změna názvu a popisu s/bez změny zdrojového dokumentu (PDF)
- *Zjištění stavu běhu robota*
- *Spuštění robota*
- *Zastavení běhu robota*

Automatické zpracování dokumentů

Pro usnadnění přidávání diplomových prací do knihovny poslouží robot (angl. Crawler) pro automatické získávání dokumentů z webových stránek. Nazveme ho FitThesisCrawler. Využijeme k tomu existující seznam všech prací[4]. Jelikož se jedná o HTML dokument, můžeme ho zpracovat pomocí nástrojů pro práci s DOM. Konkrétně se jedná o využití vyhledávání HTML elementů dle XPath výrazů. Pomocí nich získáme z HTML dokumentu kolekci odkazů na abstrakt (zvýrazněno na Obr 5-9). Ze stránky s detailem práce (viz Obr 5-7), získané pomocí adresy nalezené v předešlém kroku, pak opět s využitím XPath dotazů vyhledáme název práce, abstrakt a adresu PDF souboru s prací. Pak už jen zbývá PDF soubor z adresy stáhnout.

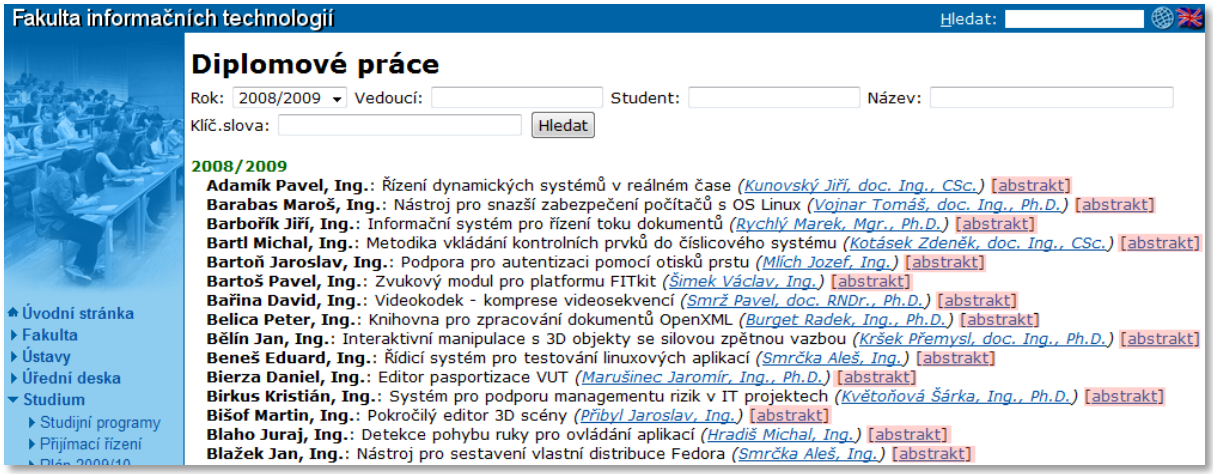
Neformální popis algoritmu vykonávaného robotem je následující:

- `Získej kolekci adres na stránky s detailem práce.`
- `Pro každou adresu z kolekce adres`
    - `Získej název, popis a adresu PDF souboru práce`
    - `Pokud některý z údajů chybí, přejdi na další adresu`
    - `Ověř, zda už práce s daným názvem existuje`

[4] http://www.fit.vutbr.cz/study/DP/DP.php?y=*

- EXISTUJE – ověř, zda je velikost souboru stejná
  - STEJNÁ – pokračuj na další adresu
  - RŮZNÁ – aktualizuj dokument přeindexováním
- NEEXISTUJE – vytvoř nový dokument

Robot FitThesisCrawler by měl běžet v samostatném vlákně a musí být zajištěna možnost jeho běh spouštět a zastavovat.



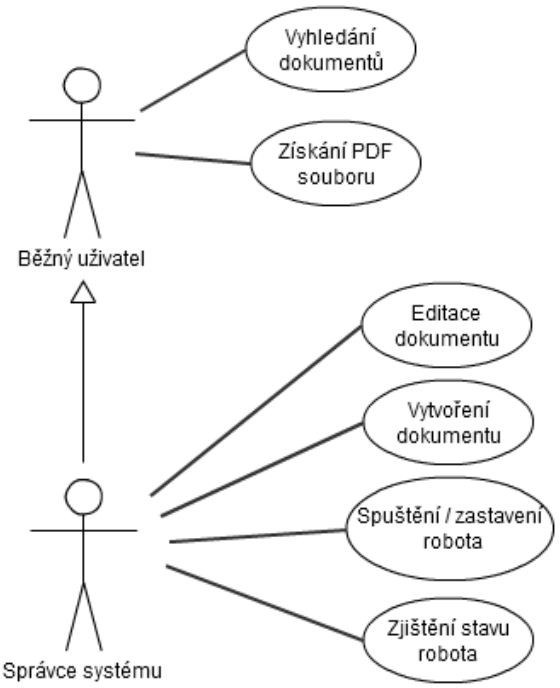
Obr 5-9 Výřez části webové stránky s rejstříkem diplomových prací

5.4.2 Tenký klient

Klient systému digitální knihovny má poskytovat uživatelské rozhraní pro přístup k dokumentům a jejich vyhledávání. K systému mohou přistupovat dvě role uživatelů – *běžný uživatel*, kterému jsou poskytnuty funkce pro vyhledávání dokumentů a získání jejich zdrojových textů v PDF, a *správce*, který navíc může editovat dokumenty a ovládat robota pro automatické získávání dokumentů. Tyto případy použití jsou znázorněny v diagramu na Obr 5-10.

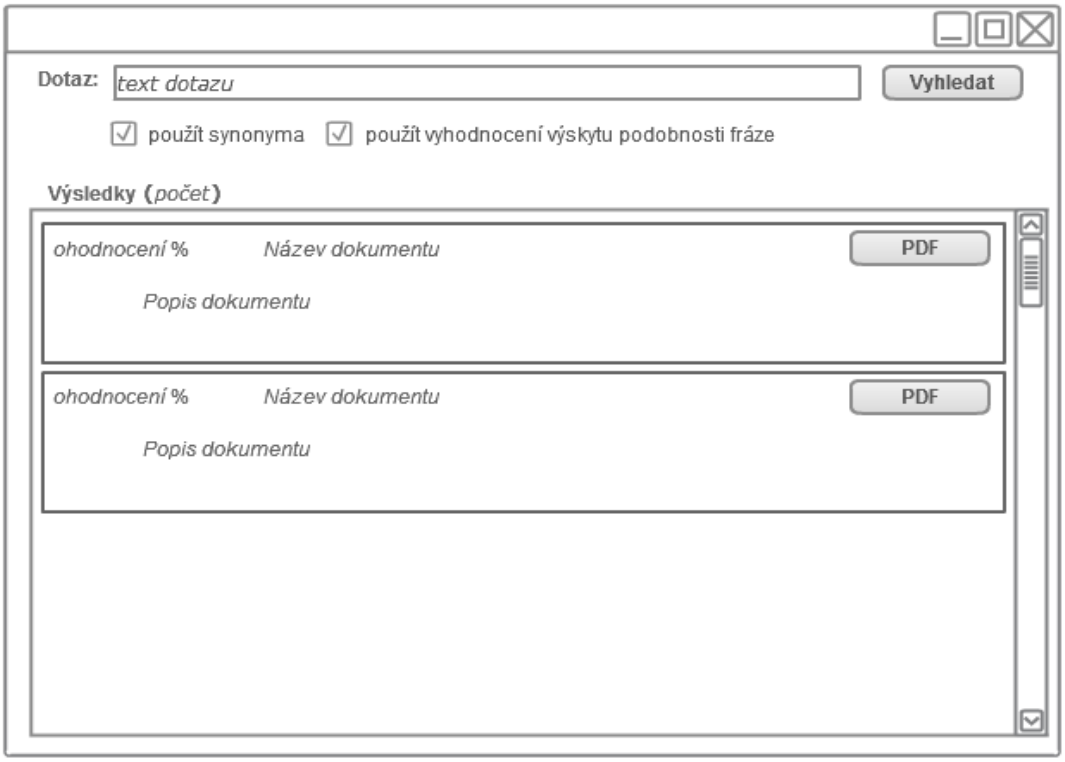
Pro komunikaci se službou knihovního systému bude využívat výše definované komunikační rozhraní. Interakce s uživatelem bude realizována prostřednictvím GUI. Uživatel bude moci zadat dotaz reprezentující jeho informační potřebu, nastavit parametry pro hledání, tedy zda se má použít rozšíření dotazu o synonyma a zda li má být využito zvýhodnění ohodnocení dokumentů obsahující podobnou strukturu slov jako dotaz (parametr *hledat frázi*). Po odeslání dotazu uživatelské rozhraní přehledně zobrazí kolekci nalezených dokumentů. Každý dokument vrácený ve výsledku vyhledávání má svoje ohodnocení, název a pro přibližnou informaci o obsahu dokumentu také popis. K získání zdrojového pdf dokumentu bude sloužit tlačítko pro uložení dokumentu na disk. Návrh GUI pro běžného uživatele je znázorněn na Obr 5-11.

Protože se jedná o aplikaci demonstrující navržený přístup k vyhledávání informací, nebude řešena žádná forma autentizace uživatelů. K rozlišení role uživatele postačí použít například parametr při spuštění aplikace.

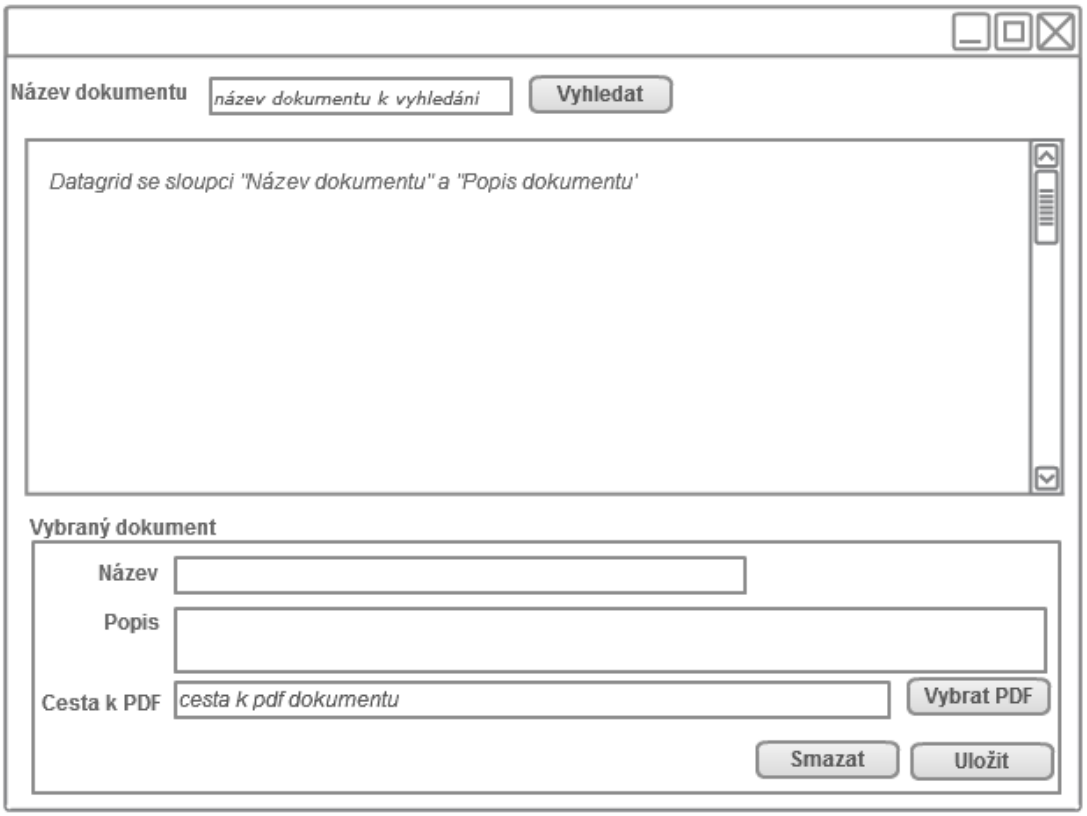


Obr 5-10 Diagram případů užití pro uživatelské rozhraní aplikace

Návrh GUI tenkého klienta:



Obr 5-11 Návrh GUI pro vyhledávání dokumentů



Obr 5-12 Návrh GUI pro správu dokumentů

6 Implementace

Cílem práce bylo vytvořit systém pro uchovávání, správu a dotazování dokumentů na příkladu elektronické knihovny diplomových prací. V předešlé kapitole „Specifikace a návrh systému“ jsem definoval chování jednotlivých částí, ze kterých se aplikace skládá. Nyní se dostáváme k popisu realizace této aplikace. K vývoji byl použit jazyk C#, tedy jedná se o aplikaci běžící nad .Net frameworkem. Pro uchování a základní dotazování dat byl využit SŘBD Postgres. Dále byly použity technologie a knihovny pro práci s PDF dokumenty, objektově relační mapování, práci s jazykovými slovníky hunspell, realizaci SOA a práci s HTML dokumenty. Tato kapitola se zabývá zejména popisem implementace klíčových částí systému.

6.1 Použité technologie a knihovny třetích stan

Fluent nHibernate

Jedná se o nadstavbu ORM frameworku nHibernate, která zjednodušuje mapování databázových tabulek na perzistentní třídy. V samotném nHibernate frameworku je mapování realizováno pomocí konfiguračních xml souborů. Jejich nevýhodou je ale přílišná „upovídanost“ a složité hledání chyb v případě špatné konfigurace. Nadstavba fluent nHibernate dovoluje zapisovat mapování prostřednictví c# kódu. Tím získáme v prostředí moderních IDE, kontrolu syntaxe a především možnost využití intellisense. Zároveň se usnadní možný refactoring a zlepší čitelnost konfigurace.

Pro samotnou práci s databází, respektive k získávání perzistentních objektů se využívá tříd zvaných DAO (Data Access Object). V nich vytváříme dotazy pomocí jazyka HQL (podobný SQL, ale umožňuje se dotazovat pomocí zápisu perzistentních tříd), kritérií (funkčně podobný HQL, ale nevytváří se textový dotaz) nebo prostřednictvím SQL dotazu pro použitý typ databáze.

Použití nHibernate frameworku usnadňuje přístup k databázi a získávání dat z ní. Díky použití proxy, která zabaluje perzistentní třídy, je možné využít opožděného načítání objektů z databáze a pohodlně pracovat s daty. Nevýhodou jeho použití je režie spojená s prací s objekty, zejména hlídání jejich stavů, a také nízká míra možnosti ovlivnit chování při zápisu do databáze. To je ale daň za vyšší míru abstrakce při práci s databází.

V ukázce níže je vidět příklad mapovací třídy pro fluent nHibernate [9]. Jak je vidět, hojně využívá lambda výrazů. Mapovací třída je poděděna od generické abstraktní třídy ClassMap, která pracuje s typem konkrétní perzistentní třídy. Voláním metody Id() se nastaví sloupec s klíčem tabulky, Map() spojí vlastnost třídy se sloupcem tabulky, References() vytvoří vztah 1:1 případně HasMany() definuje propojení 1:N.

Ukázka mapování tabulky na třídu

```
public class CatMap : ClassMap<Cat>
{
  public CatMap()
  {
    Id(x => x.Id);
    Map(x => x.Name).Length(16)
                    .Not.Nullable();
    Map(x => x.Sex);
    References(x => x.Mate);
    HasMany(x => x.Kittens);
  }
}
```

Castle Windsor Container

Poskytuje prostředí pro vytváření snadno upravovatelného, konfigurovatelného a testovatelného programů. Umožňuje použití *Inversion of Control* a *Dependency Injection*, což jsou dvě související techniky, které by se v češtině daly nazvat jako *obrácené řízení* a *vkládání závislostí*. Jedná se o přístup k programování, kdy jsou v konečném důsledku omezeny pevné vazby mezi třídami a o vzájemné propojení tříd se stará kontejner, který vytváří a spravuje jejich instance v závislosti na konfiguraci aplikace. K vlastní funkčnosti využívá reflexe.

Z praktického pohledu můžeme pomocí XML konfiguračního souboru kontejneru definovat parametry konstruktorů třídy a to jak v podobě skalárních hodnot, tak i předáním identifikátoru jiné třídy z konfigurace. Dále můžeme definovat, která třída implementující nějaké rozhraní má být v kódu použita. Následuje malá ukázka použití v implementované aplikaci.

Ukázka konfigurace komponenty

```
<properties>
    <affFilePath>data/czechDictionaries/cs_CZ.aff</affFilePath>
    <dicFilePath>data/czechDictionaries/cs_CZ.dic</dicFilePath>
    <datThesFilePath>data/czechDictionaries/th_cs_CZ_v2.dat</datThesFilePath>
</properties>
<component
      id="NHunspell"
      type="cz.vutbr.fit.xhochm01.IRCore.TextProcessing.NHunspell,
cz.vutbr.fit.xhochm01.IRCore"
      service="cz.vutbr.fit.xhochm01.IRCore.TextProcessing.IStemmer,
cz.vutbr.fit.xhochm01.IRCore">
      <parameters>
        <affFilePath>#{affFilePath}</affFilePath>
        <dicFilePath>#{dicFilePath}</dicFilePath>
        <datThesFilePath>#{datThesFilePath}</datThesFilePath>
      </parameters>
 </component>
```

Ukázka získání instance komponenty

```
IStemmer stemmer = (IStemmer)container[typeof(IStemmer)];
```

Castle framework také usnadňuje integraci ORM nHibernate do aplikace a to díky poskytovaným facility, které se starají o správu a poskytování session sloužících k dotazování se.

Windows Communication Foundation

Jedná se o součást .Net frameworku verze 3.0, která slouží k vytváření na služby orientovaných aplikací . Windows Communication Foundation (WCF) je navržen v souladu s principy SOA (Service Oriented Architecture) pro podporu distribuovaných výpočtů, kde jsou poskytované služby využívány spotřebiteli (klienty) [10]. Klienti mohou využívat více služeb a služba může být využívána více klienty. Služby WCF poskytují komunikační rozhranní, které klienti využívají, bez ohledu na to, na jaké platformě je služba hostována. Pro komunikaci WCF nabízí několik kanálů, od jednoduchého spojení, po pokročilé webové služby.

WCF služba se skládá ze tří částí. Z třídy Service, implementující službu poskytovanou komunikačním rozhranním, hostitelského prostředí, které hostí služby, a koncových bodů (Endpoints), prostřednictvím nichž se budou klienti připojovat ke službě. Veškerá komunikace ve WCF probíhá prostřednictvím koncových bodů. Ty definují dohodu (Contract) specifikující, které metody třídy služby budou jejich prostřednictvím zpřístupněny klientům. Každý z koncových bodů může definovat jinou množinu metod, které zpřístupní. Koncové body také specifikují spojení (Binding), pomocí kterého se bude klient ke službě připojovat, a adresu, na které je hostovaná služba dostupná. WCF nabízí několik předdefinovaných spojení pro nepoužívanější komunikační protokoly, jako jsou „SOAP over HTTP“, „SOAP over TCP“, „SOAP over Message Queues“ atp. Samotné služby mohou být hostovány na IIS (webový aplikační server od Microsoftu), implementovány jako Windows služba nebo jako samostatná konzolová nebo GUI aplikace.

Ukázka nastevení služby pomocí konfiguračního souboru *app.config*:

```
<system.serviceModel>
 <services>
  <service name="cz.vutbr.fit.xhochm01.LibrarySearch.Service.LibrarySearchService">
   <endpoint address="net.tcp://192.168.2.2:8082/IRSearch"
             binding="netTcpBinding"
             bindingConfiguration="LibrarySearchService"
             contract="cz.vutbr.fit.xhochm01.LibrarySearch.Shared.ILibrarySearchService"
             name="cz.vutbr.fit.xhochm01.LibrarySearch.Shared.ILibrarySearchService" />
  </service>
 </services>
 <bindings>
  <netTcpBinding>
   <binding name="LibrarySearchService" receiveTimeout="06:00:00" sendTimeout="06:00:00">
    <security mode="None">
     <transport clientCredentialType="Windows" />
    </security>
   </binding>
  </netTcpBinding>
 </bindings>
</system.serviceModel>
```

PDF Box

Soubory dokumentů PDF jsou známy svou složitostí interpretace jejich obsahu. Pro funkčnost demonstrační aplikace je potřeba z PDF dokumentů získat čistý text. Jednou z knihoven pro práci s PDF a mimo jiné i extrakci textu z nich je právě PDF Box. Ten je sice napsaný v javě, ale díky knihovně IKVM je možné její použití i v prostředí .Net aplikací. Původně jsem zamýšlel použít čistě .Net knihovny, ale po jejich podrobném studiu jsem nedospěl k uspokojivému výsledku při potřebě extrakce textu. Samotný proces získání textu totiž není jednoduchý, protože obsahem PDF dokumentu jsou objekty rozmístěné po stránkách. Nelze tak snadno určit posloupnost textů či znaků tak, aby odpovídala podobě, jakou můžeme vidět ve čtečkách PDF.

HTML Agility Pack

Pro potřeby FitThesisCrawler robota je potřeba pokládat XPath dotazy nad DOM HTML dokumentu. V .Net frameworku sice existuje třída XMLDocument, která by na první pohled mohla postačovat, problém však je ve zpracování onoho HTML dokumentu. Ten totiž není správně strukturovaný (well formed) dle specifikace XML. Knihovna HTML Agility Pack však umí zpracovat HTML dokumenty a díky ní můžeme vyhledávat pomocí XPath výrazů.

nHunspell

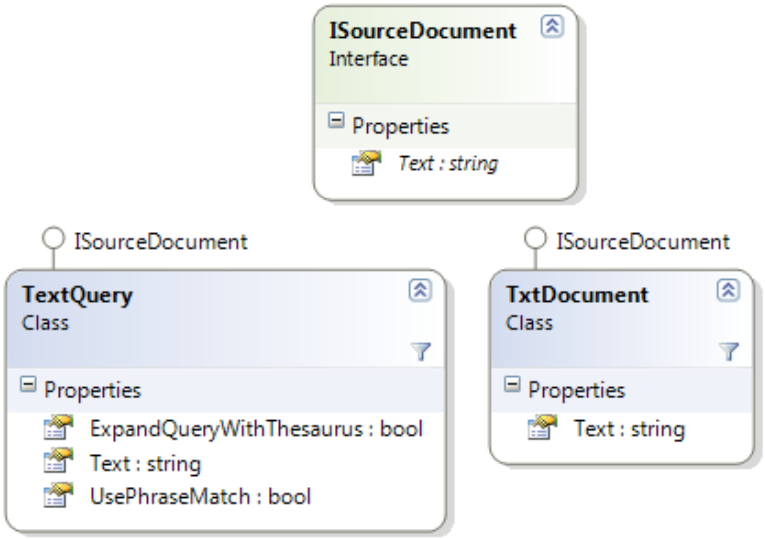
Umí pracovat se slovníky hunspell a z pohledu potřeb vyhledávacího jádra poskytuje rozhraní pro stemming slov a získávání synonym. Struktura slovníků, se kterými pracuje, a jeho přibližná funkce byla popsána v kapitole 5.3.2 v části o stemmingu a hledání synonym.

6.2 Vyhledávací jádro

Vyhledávací jádro je implementováno jako samostatná knihovna. Z tohoto důvodu je možné ho využít i v jiných aplikacích mimo tu demonstrační. Jeho specifikace a návrh byly popsány v kapitole 5.3. Pro práci s databází je využit výše představený Fluent nHibernate. Dle navržených entit byly vytvořeny perzistentní třídy a podle databázového schéma bylo nastaveno mapování pro každou z nich.

Práce s dokumenty, tedy jejich indexování, mazání a vyhledávání informací v nich, se děje prostřednictvím tříd *DocumentProcesser* (indexace a mazání) a *Searcher* (pro ohodnocené vyhledávání). Tyto dvě hlavní třídy jsou popsány níže.

Dokumenty a dotazy jsou reprezentovány třídami *TxtDocument* a *TextQuery*, které mají společné rozhranní *ISourceDocument* (viz. Obr 6-1) poskytující textový řetězec. S tímto rozhranním pak pracuje třída *TermStringCreator*, která zajišťuje zpracování textu, tedy získání termů a jejich pozic. Tento proces je navržen v kapitole 5.3.2 a dle tohoto návrhu jej třída implementuje.



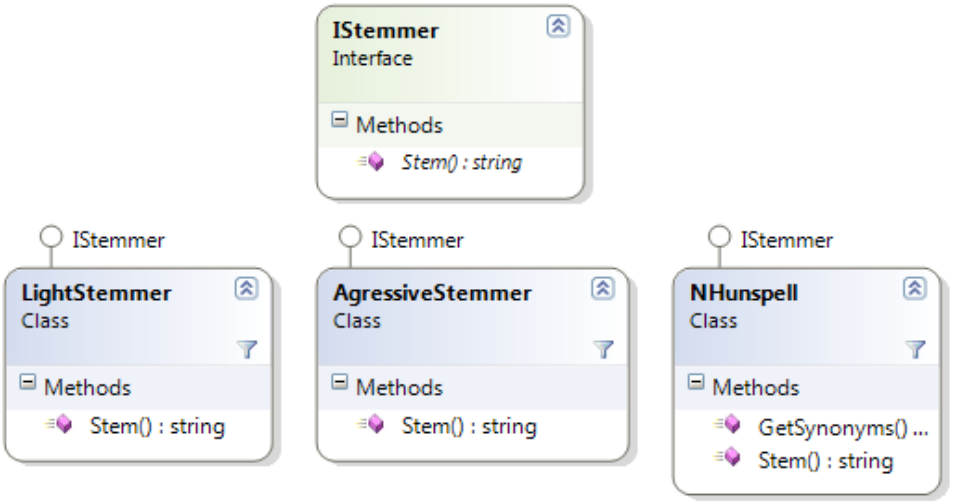
Obr 6-1 třídy pro vstupní data - dokument a dotaz - implementující spolenčné rozhranní

Stemmery

Aplikace implementuje 3 typy stemmerů. Analytický lehký a agresivní typ a slovníkový s využitím knihovny hunspell. Tyto stemmery implementují stejné rozhraní *IStemmer* a je tak možná jejich snadná záměna (Obr 6-2). Ta je navíc možná pouhou změnou konfigurace aplikace, tedy bez nutnosti měnit kód a program znovu kompilovat. Tato změna se provádí v souboru *Container.config*. nastavením parametru „service“ na hodnotu:

```
service="cz.vutbr.fit.xhochm01.IRCore.TextProcessing.IStemmer,
         cz.vutbr.fit.xhochm01.IRCore"
```

u vybrané komponenty reprezentující konkrétní stemmer. Vybraný stemmer je poté využívám ve výše uvedené třídě *TermStringCreator*.



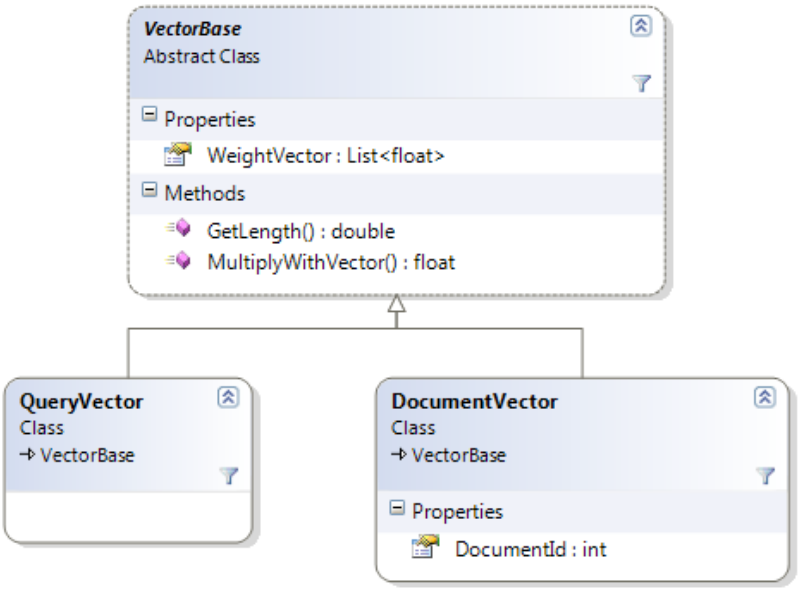
Obr 6-2 Třídy stemmerů implementující stejné rozhranní IStemmer

Třída DocumentProcesser

DocumentProcesser, konkrétně jeho metoda *IndexDocument* s parametrem typu *ISouceDocument*, slouží k indexaci dokumentu. To je provedeno dle návrhu z kapitoly 5.3.3. Dále třída poskytuje metodu *DeleteDocument*, která umožňuje odstranění naindexovaného dokumentu. Obě metody využívají stejného zámku, zajišťující výlučný přístup k úpravám struktury dokumentů, aby nedošlo k nekonzistenci mezi daty v databázi a těmi načtenými a upravovanými v aplikaci.

Třída Searcher

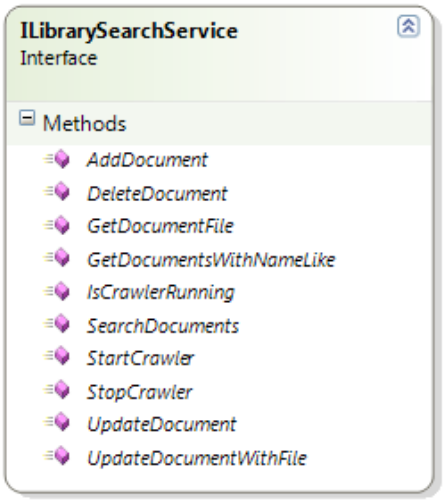
Tato třída slouží k vyhledávání informací v dokumentech a vrací kolekci ohodnocených výsledků reprezentovaných třídou *SearchResult*. Její implementace se řídí návrhem z kapitoly 5.3.3. Pro reprezentaci termů v dokumentech při vyhledávání je sice možné použít perzistentní třídy, ale ty mají v případě oboustranné viditelnosti mnohonásobná vzájemná propojení a jejich použití v procesu zpracování je neefektivní. Proto je vhodné do databáze pokládat skalární dotazy a potřebné hodnoty reprezentovat specifickými třídami. K tomu posloužily třídy *DocumentTermSkeleton* a *QueryTermSkeleton*, které reprezentují term v dokumentu a dotazu, a udržují hodnoty TF, IDF, případně pozice termu v textu, a v případě *QueryTermSkeleton* i hodnoty synonym. Po jejich volitelném zpracování, formou agregace termů vyskytujících se v dotazu a jejich synonym, z nich jsou vypočteny vektory vah. To je realizováno dle tf-idf schéma vektorového modelu, viz. kapitola 2.3. Tyto vektory reprezentují třídy *DocumentVector* a *QueryVector* (Obr 6-3), které umožňují provádět operace potřebné pro výpočet podobnosti ve vektorovém modelu (rovnice 2.14). Po výpočtu podobnosti dokumentu a dotazu s použitím vektoru vah, dochází k volitelné korekci ohodnocení dle podobnosti výskytu fráze. K tomu slouží třída *PhrasesSimilarityComputer* pracující s instancemi *QueryTermSkeleton* a *DocumentTermSkeleton*. Její implementace je dána výpočtem uvedeným v kapitole 2.4.2. Výsledkem je kolekce objektů třídy *SearchResult*, která je před vrácením vyfiltrována o položky s ohodnocením nižším, než je definovaná mez.



Obr 6-3 Třídy reprezentující váhový vektor dotazu a dokumentu se společným abstraktním předkem

6.3 Demonstrační aplikace

Jak již bylo několikrát zmíněno, demonstrační aplikace je navržena na bázi SOA a k tomu využívá WPF. Komunikační rozhranní pro službu a klienta definuje je definováno rozhranním *ILibrarySearchService* (Obr 6-4). To je pro potřeby WPF označeno atributem „ServiceContract“ a metody jím poskytované atributem „OperationContract“, podle kterých rozpozná, co ono komunikační rozhranní tvoří. Toto rozhraní je společně s třídami, použitými ve službě a klientovi, umístěno do samostatné knihovny.



Obr 6-4 Rozhranní definující metody poskytované službou

6.3.1 Serverová část

Serverovou část aplikace tvoří knihovní vyhledávací služba, která implementuje metody komunikačního rozhraní třídou *LibrarySearchService*. Tato třída má nastaveny parametry, které definují chování hostované služby:

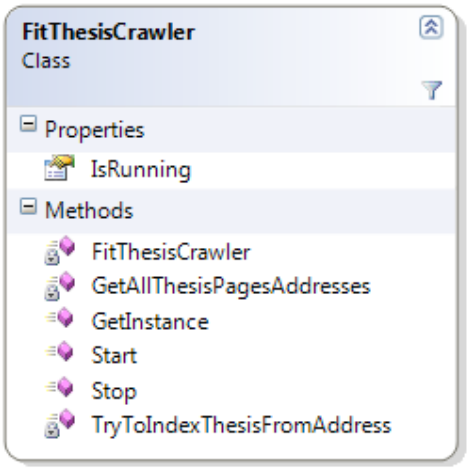
```
ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)
```

Konkrétně toto nastavení specifikuje vytváření instancí samotné služby na jednu po celé trvání spojení s klientem.

Text dokumentů určený ke zpracování vyhledávacím jádrem je získáván z PDF dokumentů využitím knihovny PDFBox. Tyto dokumenty jsou následně ukládány na disk do adresářů pojmenovaných dle náhodného GUID a cesta k nim je uložena do databáze.

Knihovní vyhledávací služba se dále stará o běh robota pro automatickou indexaci dokumentů z webu. Jeho požadované chování bylo popsáno v kapitole 5.4.1. Je implementován třídou *FitThesisCrawler* (Obr 6-5), která je implementována jako singleton. Samotný proces automatické indexace běží v samostatném vláknu a je možné ho voláním metod *Start()* a *Stop()* spustit či zastavit.

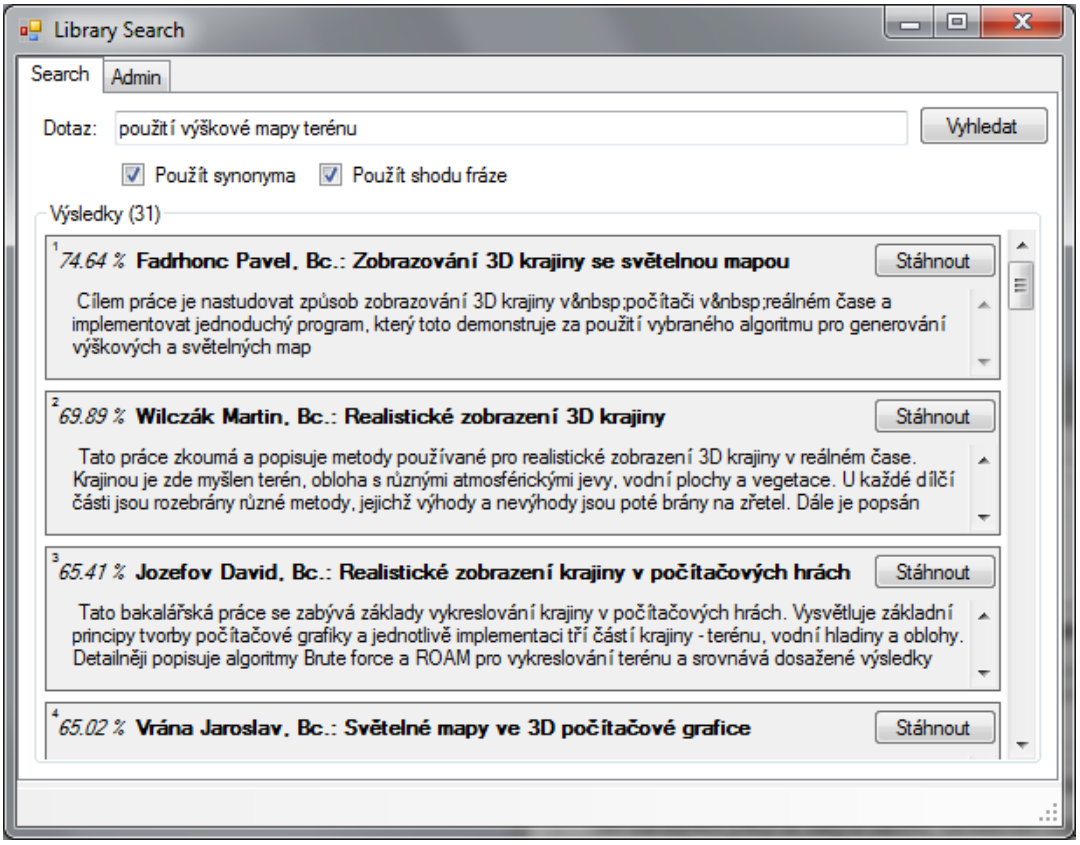
Konfigurace komunikace služby je definována v souboru „app.config“, kde je možné nastavit ip adresu a port, na které služba naslouchá příchozím spojením. Nastavení chování vyhledávacího jádra je možné změnit úpravou souboru „container.config“. Konkrétně se jedná o změnu cest k souboru se seznamem stop slov, slovníkům hunspell a také změnu použitého stemmeru, která již byla popsána výše.



Obr 6-5 Třída robota pro automatickou indexaci dokumentů z webu

6.3.2 Tenký klient vyhledávací služby

Klientská část aplikace slouží pouze k volání služby a zobrazování obdržených výsledků. Pro komunikaci se službou slouží třída *LibrarySearchClient*, která implementuje již výše zmíněné rozhranní *ILibrarySearchService* a dědí od generické *ClientBase*. Díky tomu mohou být ve třídě volány metody implementované službou. Nastavení komunikace je opět prostřednictvím souboru „app.config“. Pro rozlišení rolí běžný uživatel a administrátor, se aplikace spouští s a nebo bez parametru „admin“. Část vytvořeného uživatelského rozhranní je vidět na Obr 6-6.



Obr 6-6 GUI klientské aplikace - část určená pro vyhledávání

6.4 Použití aplikace na unix systémech

Implementovaná aplikace je určena primárně pro systémy windows s nainstalovaným .Net frameworkem ve verzi 3.5 a vyšší. Pro systémy unix existuje mono framework, který poskytuje prostředí pro běh aplikací napsaných pod .Net. V současné době je implementována podpora aplikací v .Net ve verzi 1.1 a 2.0 plně a ve verzích 3.0 a 3.5 částečně.

Při testování vyvinuté aplikace v unixu nastal problém s neúplnou implementací WCF, kdy jsou ignorovány nastavení týkající se velikosti přenášených zpráv mezi službou a klientem. Je tak použita napevno nastavená velikost 64kB, čímž vzniká problém při přenosu PDF souborů, případně jiných serializovaných dat, jako jsou například výsledky vyhledávání, které by překročili tuto hodnotu.

Dále použitá knihovna nHunspel používá částečně nativní dll knihovnu, bez které nefunguje stemming slov. Je možné ji tak možné použít pouze pro vyhledávání synonym, jelikož thesaurus je napsaný v čistém .Net kódu.

Posledním problémem na který jsem narazil, ale který se dá vyřešit, je ignorování parametru probing v souboru „app.config“, který definuje cesty k nestandardním knihovnám použitým v kódu. Mono tak při spouštění aplikace nenajde požadované knihovny a aplikace se nespustí. To lze obejít nakopírováním všech knihoven z přehledné struktury podadresářů do složky s aplikací.

7 Experimenty

V této kapitole se seznámíme s experimenty na implementovaném systému. Bude provedena série testů nad zvolenými daty, pomocí nichž získáme srovnání vlivu použitého typu stemmeru na efektivitu vyhledávání. Dále uvidíme přínos rozšíření dotazu o synonyma a použití algoritmu pro zvýhodnění ohodnocení dokumentů dle podobnosti výskytu fráze z dotazu.

7.1 Testovací data

Pro zhodnocení efektivity systému pro vyhledávání informací se využívá obsáhlých kolekcí dokumentů nazývaných korpusy. Pro tyto dokumenty jsou navrženy dotazy, kde každý dotaz má danou množinu relevantních dokumentů pro konkrétní korpusy.

Dostupné korpusy, např. na webu organizace TREC[5], bohužel povětšinou obsahují dokumenty v angličtině nebo v jiných cizích jazycích. Implementovaný systém je navržený pro práci s dokumenty v českém jazyce, tyto kolekce dokumentů jsou tedy v našem případě nepoužitelné.

S ohledem na implementovaný systém, tedy aby se otestoval celý systém jako celek, jsem se rozhodl použít korpus dokumentů složených ze všech bakalářských prací dostupných na webu FIT VUT v Brně. Jedná se o kolekci 680 technicky zaměřených textů. Pro tyto dokumenty jsem navrhl několik dotazů (Tabulka 7-1) a pro každý z nich letmým přezkoumáním stanovil množinu relevantních dokumentů.

Identifikátor	Text dotazu
Q1	Výšková mapa terénu
Q2	Operace nad datovou kostkou
Q3	Použití datových skladů
Q4	Zjednodušení 3d modelu
Q5	Multiagentní systémy

Tabulka 7-1 Navržené dotazy pro korpus bakalářských prací

7.2 Vliv stemmeru na efektivitu vyhledávání

Demonstrační aplikace využívá, v závislosti na nastavení jeden z tří možných stemmerů. Pro zjištění, který z nich je pro využití nejvhodnější, provedeme sérii testů nad výše specifikovanými daty. Budeme měřit úplnost a přesnost celého vyhledávacího systému.

[5] http://trec.nist.gov/data.html

Po naindexování všech dokumentů z korpusu můžeme pozorovat první vliv stemmeru na dokumenty a tím je velikost slovníku, respektive průměrný počet termů v dokumentu (Tabulka 7-2).

Typ použitého stemeru	Průměrný počet termů na dokument
Hunspell stemmer	1982
Light stemmer	1897
Agressive stemmer	1757

Tabulka 7-2 Vliv stemmeru na počet termů v dokumentech

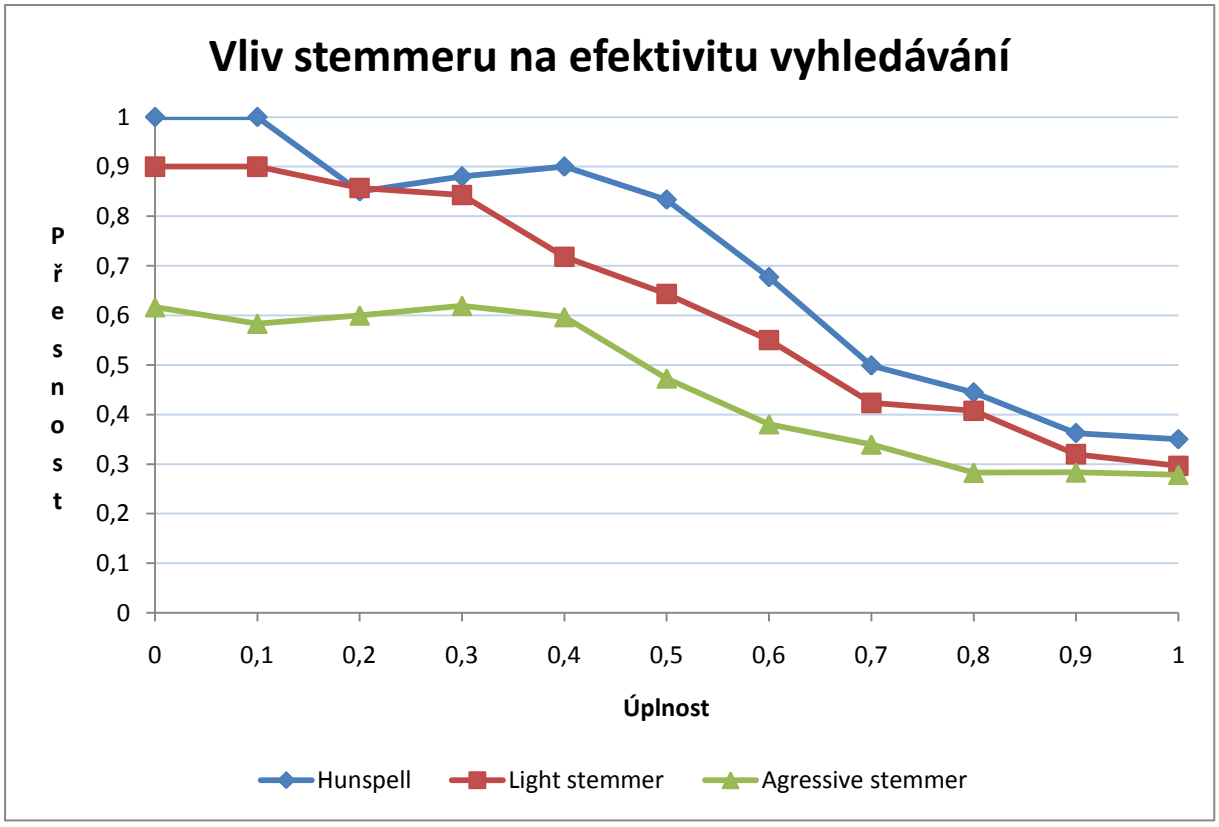
Vyšší počet termů může znamenat výskyt dvou a více slov se stejným základem, ale stále jiným tvarem a což může zhoršit schopnost správného vyhodnocení dokumentu. A naopak nižší počet může značit účinné převedení slov do základního tvaru a tím pádem šanci nalézt více relevantních dokumentů. Z druhého pohledu tomu ale může být naopak a nižší počet termů znamená, že se dvě významově jiná slova převedla na stejný tvar. To by mělo za důsledek výskyt vyššího počtu falešných pozitiv při vyhledávání informací v dokumentech.

Průběh testování

Výše definované dotazy byly postupně vyhodnocovány pro každý typ stemmeru a byly zaznamenávány pozice relevantních dokumentů. Minimální hodnota podobnosti pro výsledky byla stanovena na hodnotu 30%. Aby bylo možné mezi sebou výsledky porovnávat, byly výsledky vyhodnocení pro každý dotaz přepočítány do 11 úrovní přesnosti. Po proběhnutí všech dotazů byly výsledky pro každý typ stemmeru zprůměrovány a zobrazeny v tabulce 7-3.

Úplnost	Přesnost		
	Hunspell stemmer	Light stemmer	Agressive stemmer
0	1	0,9	0,62
0,1	1	0,9	0,58
0,2	0,85	0,86	0,59
0,3	0,88	0,84	0,62
0,4	0,9	0,72	0,6
0,5	0,83	0,64	0,47
0,6	0,68	0,55	0,38
0,7	0,5	0,42	0,34
0,8	0,44	0,41	0,28
0,9	0,36	0,32	0,28
1	0,35	0,3	0,27

Tabulka 7-3 Výsledky testování efektivity vyhledávání při použití různých stemmerů



Obrázek 7-1 Vliv stemmeru na efektivitu vyhledávání

	Hunspell stemmer	Light stemmer	Agressive stemmer
MAP	0,696	0,621	0,464

Obrázek 7-2 MAP pro použití jednotlivých stemmerů

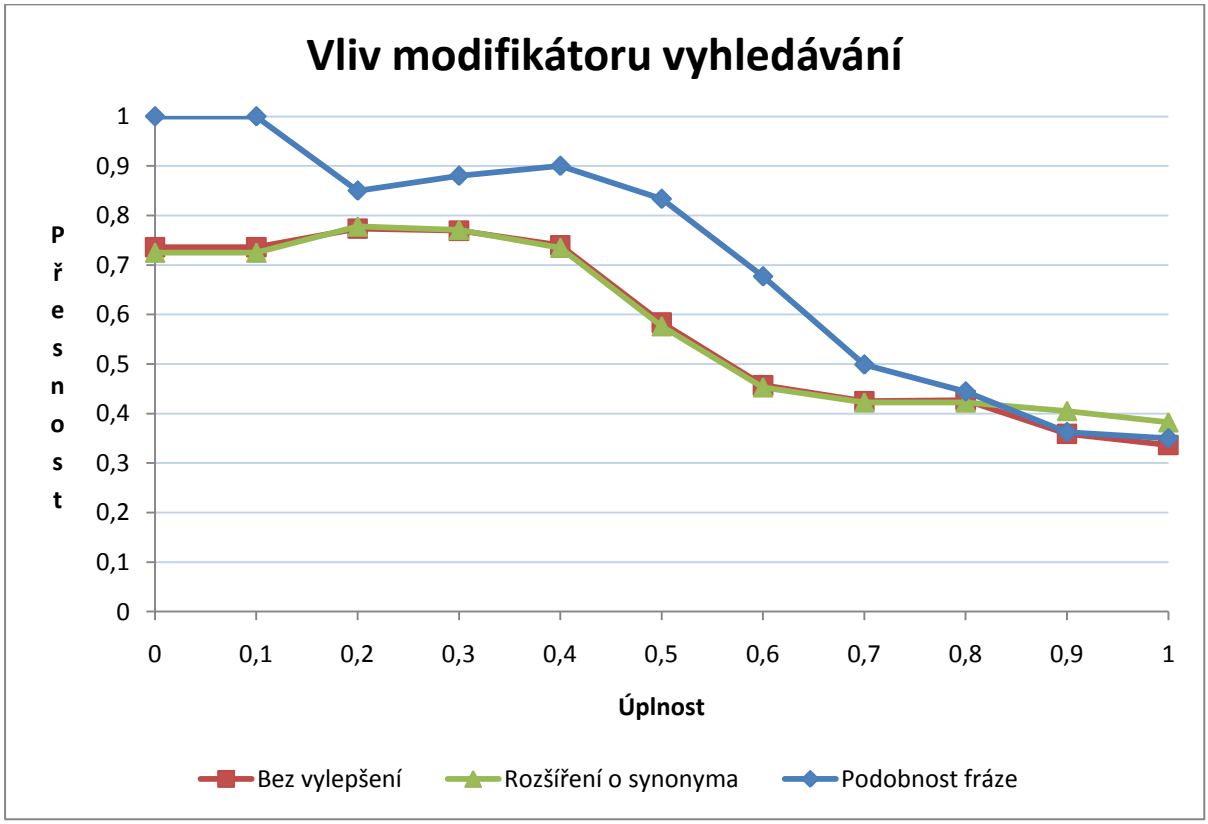
Z křivky přesnost-úplnost je vidět značný negativní vliv agresivního stemmeru na efektivitu vyhledávání. Nastala ta horší situace z výše napsaných, a to, že nižší počet termů na dokument znamená použití stejné hodnoty termu pro 2 různá slova. Došlo tedy k přeměně slova na příliš obecný tvar. Naopak stemmer realizovaný pomocí hunspell slovníku vychází ve výsledku nejlépe.

7.3 Vliv vylepšení vyhledávání na efektivitu

V demonstrační aplikaci je při vyhledávání možné zvolit použití rozšíření dotazu o synonyma a použití modifikace ohodnocení dokumentů podle podobnosti výskytu fráze z dotazu v něm. Systém vyhledávání využívá hunspell stemmer, jelikož jeho výsledky byly v předešlém testu nejlepší. Testování proběhlo, stejně jako v minulém případě, na zvolené kolekci bakalářských prací a s použitím definované kolekce dotazů.

Úplnost	Přesnost		
	Bez modifikátorů	Rozšíření o synonyma	Podobnost fráze
0	0,74	0,73	1
0,1	0,74	0,73	1
0,2	0,77	0,78	0,85
0,3	0,76	0,77	0,88
0,4	0,74	0,73	0,9
0,5	0,58	0,58	0,83
0,6	0,46	0,45	0,68
0,7	0,43	0,42	0,5
0,8	0,43	0,42	0,44
0,9	0,36	0,41	0,36
1	0,34	0,38	0,35

Tabulka 7-4 Výsledek testování různých nastavení vyhledávání



Obrázek 7-3 Vliv použitých volitelných modifikátorů vyhledávání na jeho efektivitu

	Bez modifikátorů	Rozšíření o synonyma	Podobnost fráze
MAP	0,696	0,53	0,57

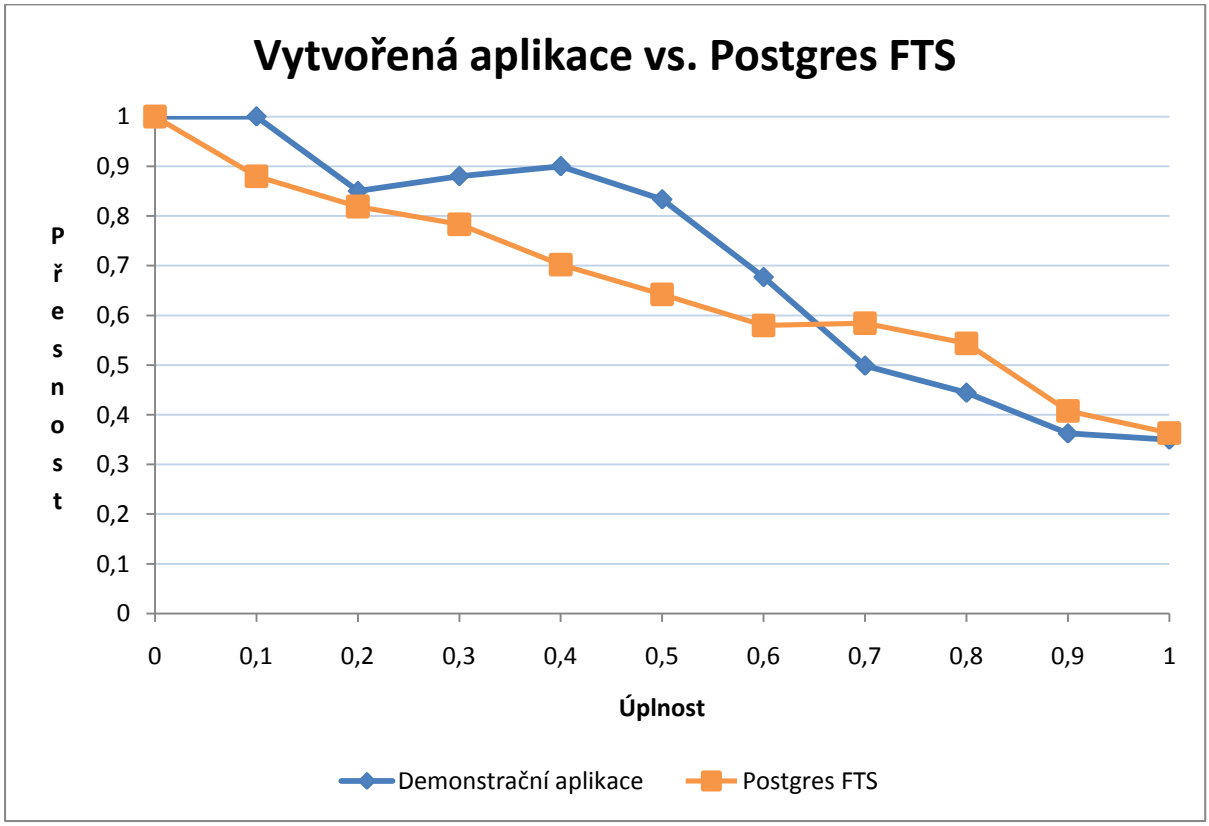
Tabulka 7-5 MAP pro modifikace vyhledávání

Z výsledků vyhodnocení efektivity vyhledávání je vidět značný nárůst přesnosti při využití modifikace pomocí vyhledávání podobnosti fráze. Naopak při rozšíření dotazu o synonyma nedochází k žádnému, případně jen minimálnímu, nárůstu přesnosti. Je to především způsobeno absencí synonym pro slova z většiny dotazů. V případě, že se synonyma vyskytovala, došlo ve většině případů pouze k malému nárůstu ohodnocení, které nemělo vliv na pořadí výsledků. To jen potvrdilo známý fakt, že rozšíření dotazu o synonyma ve většině případů nepřináší výraznou změnu a může spíše způsobit snížení ohodnocení relevantních dokumentů.

7.4 Porovnání s Postgres FTS

Databáze Postgres umožňuje fulltextové vyhledávání prostřednictvím open source modulu TSearch2. Tento modul používá pro reprezentaci dokumentů datový typ tsvector a pro dotazy tsquery. Má zabudovaný tokenizer, který tokeny rozděluje několika typů, podle toho z jakých znaků se skládají. Pro každý typ je možné definovat použití jiného stemmeru. Samotný stemming je řešený buď použitím analytického snowball, nebo využitím slovníků ispell.

Pro možnosti porovnání byl TSearch2 v databázi nakonfigurován s využitím českého ispell slovníku a seznamu českých stop slov. Databáze byla poté naplněna stejnou kolekcí dokumentů jako demonstrační aplikace a byly v ní vyhledány dokumenty pro celou kolekci testovacích dotazů. Demonstrační aplikace využívala vyhledávání za použití rozšíření dotazu o synonyma a zvýhodňování dokumentů dle podobnosti výskytu fráze. Výsledky testování byly vyneseny do grafu (Obr 7-4) znázorňující křivky přesnost-úplnost vytvořené aplikace a Postgres FTS.



Obr 7-4 Porovnání efektivity vyhledávání vytvořené aplikace a Postgres FTS

	Vytvořená aplikace	Postgres FTS
MAP	0,696	0,67

Tabulka 7-6 MAP pro Vytvořenou aplikaci a vyhledávání v Postgres

V porovnání s několik let vyvíjeným systémem TSearch2 pro Posgres vychází vytvořená aplikace pro vyhledávání v digitálních knihovnách poměrně dobře. Její ohodnocení sice není výrazně vyšší (viz. Tabulka 7-6), ale v porovnání s profesionálním systémem pro vyhledávání lze i takto malý nárůst považovat za úspěch.

7.5 Rychlost aplikace

K uskutečněným testům by bylo vhodné zmínit rychlost, s jakou aplikace vyhledává dokumenty. Tu je totiž možno považovat za Achillovu patu celého systému. Jelikož indexovaný obsah dokumentu je uložen v relační databázi bez využití speciálních datových typů, je nutné při indexování do databáze vkládat a aktualizovat řádově tisíce až jednoty desetitisíců řádků. To je však časově náročný proces, nejen z důvodu velkého množství komunikace mezi SQL serverem a aplikací, a například aktualizace IDF termů v testovací kolekci dokumentů trvala řádově desítky vteřin. Při vyhledávání jsou z databáze získávány hodnoty NTF a IDF pro termy ze všech dotazem zasažených dokumentů. V testovací kolekci pak výsledek dotazu do databáze obsahoval více než milion řádků. Rychlost vyhledávání dokumentů se tak pro testovací korpus pohybovala v průměru kolem 45 sekund. Aplikace běžela na osobním počítači se systémem Windows 7 a konfigurací s procesorem AMD Athlon X2 5400 a 4GB RAM.

8 Závěr

Cílem této práce bylo seznámit se s problematikou vyhledávání informací, zejména pak modely k tomu určenými a s nimi spojenými metodami pro hodnocení systémů pro vyhledávání informací. V souvislosti s modely pro vyhledávání jsem se věnoval způsobům jak zohlednit výskyt frází z dotazu při vyhledávání dokumentů, kde jsem prezentoval teorii ohodnocení dokumentů dle podobnosti výskytu fráze dané dotazem. Dále jsem se věnoval rozborů principů zpracování textu pro IR systémy s ohledem na efektivitu vyhledávání informací, tedy způsob tokenizace, eliminace často se vyskytujících slov a získání základního tvaru slova různými metodami. S tím související je i použití slovníku synonym pro rozšíření dotazů.

V druhé části práce jsem se zabýval návrhem systému pro vyhledávání s využitím vektorového modelu, rozšíření dotazu o synonyma a určení míry podobnosti výskytu fráze v dokumentech, vycházející z prezentované teorie. Tento systém jsem implementoval na .Net platformě v podobě aplikace pro vyhledávání v diplomových pracích.

Zhodnocením efektivity vyhledávání se zabývá kapitola 7. Z testů v ní popsaných vyplývá, že je výhodnější použít pro stemming slov slovníkový stemmer. Dále je vidět minimální vliv rozšíření dotazu o synonyma na výsledky vyhledávání. Úspěchem bylo použití modifikace ohodnocení dokumentu pomocí míry podobnosti výskytu fráze z dotazu v dokumentu. V poslední řadě bych zmínil nízkou rychlost systému, zapříčiněnou hlavně způsobem uložení naindexovaného obsahu dokumentu. Zároveň s tím je ale třeba poznamenat, že práce nebyla zaměřena na rychlost.

Směr dalšího výzkumu a vývoje této práce by se mohl týkat návrhu a implementace z pohledu rychlosti efektivnějšího uložení indexovaného obsahu dokumentů. Mohlo by se jednat o využití specializovaných datových typů, které poskytuje například Postgres (tsvector). Dále by bylo užitečné, kdyby docházelo k rozpoznání jazyka, ve kterém je dokument napsaný. Poté by tyto dokumenty mohly být rozděleny do skupin dle jazyka a vyhledávání by probíhalo ve skupinách dokumentů v závislosti na jazyku dotazu. Pro každý jazyk by se tak používaly specifické stemmery, seznamy stop slov atd. Poté by se už přímo vybízelo vytvořit systém, který by umožňoval vícejazyčné vyhledávání. Tím by bylo možné vyhledat pomocí jednoho dotazu relevantní dokumenty napříč různými jazyky.

Literatura

[1] BAEZA-YATES, Ricardo - RIBEIRO-NETO, Berthier. *Modern information retrieval*. New York: ACM Press, 1999. 513 s. ISBN 0-201-39829-X

[2] MANNING, Christopher D., RAGHAVAN, Prabhakar, SCHÜTZE, Hinrich. *An Introduction to Information Retrieval*. New York: Cambridge University Press, 2008. 496 s. 1. edice. ISBN 978-0-521-86571-5.

[3] PÁNEK, Karel. Architektury a modely webových strojů. *LUPA : server o českém internetu* [online]. 2002 [cit. 2009-12-28]. Dostupný z WWW: <http://www.lupa.cz/clanky/architektury-a-modely-webovych-stroju/>.

[4] PÁNEK, Karel. Šrotujeme text. *LUPA : server o českém internetu* [online]. 2002 [cit. 2009-12-30]. Dostupný z WWW: <http://www.lupa.cz/clanky/srotujeme-text/>.

[5] RIJSBERGEN, C.J. van. *INFORMATION RETRIEVAL* [online]. 1999 [cit. 2009-12-28]. Dostupný z WWW: <http://www.dcs.gla.ac.uk/~iain/keith/index.htm>.

[6] SILVA, Ilmério R., SOUZA, João Nunes, SANTOS, Karina S. . Dependence Among Terms in Vector Space Model. In *Database Engineering and Applications Symposium, International*. Los Alamitos, CA, USA : IEEE Computer Society, 2004. s. 97-102. Dostupný z WWW: <http://doi.ieeecomputersociety.org/10.1109/IDEAS.2004.1319782>. ISSN 1098-8068.

[7] WIKIPEDIA, Contributors. *Dot product* [online]. Wikipedia, The Free Encyclopedia, 2001-2009 [cit. 2009-12-26]. Dostupný z WWW: <http://en.wikipedia.org/w/index.php?title=Dot_product&oldid=332835230>.

[8] WIKIPEDIA, Contributors. *Information retreival* [online]. Wikipedia, The Free Encyclopedia, 2001-2009 [cit. 2009-12-29]. Dostupný z WWW: <http://en.wikipedia.org/w/index.php?title=Information_retrieval&oldid=334215389>.

[9] Fluent NHibernate, Contributors. *Introduction* [online]. Fluent NHibernate wiki [cit. 2010-5-5]. Dostupný z WWW: <http://wiki.fluentnhibernate.org/index.php?title=Getting_started&oldid=294>.

[10] WIKIPEDIA, Contributors. *Windows Communication Foundation* [online]. Wikipedia, The Free Encyclopedia, 2001-2010 [cit. 2010-05-17]. Dostupný z WWW: < http://en.wikipedia.org/w/index.php?title=Windows_Communication_Foundation&oldid=362573507 >.

Seznam příloh

Příloha 1: CD s elektronickou verzí technické zprávy a demonstrační aplikací.